Methodology for Integrating the Scenario Databases of Simulation Systems

THESIS

Emília M. Colonese
Captain, Brazilian Air Force

AFIT/GCS/ENG/99J-03

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/99J-03

Methodology for Integrating the Scenario Databases of Simulation Systems

THESIS

Emília M. Colonese
Captain, Brazilian Air Force

AFIT/GCS/ENG/99J-03

AFIT/GCS/ENG/99J-03

Methodology for Integrating the Scenario Databases of Simulation Systems

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Computer Systems

Emília M. Colonese
Captain, Brazilian Air Force

June 1999

AFIT/GCS/ENG/99J-03

Methodology for Integrating the Scenario Databases of Simulation Systems

Emília M. Colonese
Captain, Brazilian Air Force

Approved:

Thomas C. Hartrum — 14 May 99

Dr. Thomas C. Hartrum
Committee Chair — date

Henry B. Potoczny — 14 May 99.

Dr. Henry B. Potoczny
Committee Member — date

[illegible] — 14 May 99

Maj. Michael L. Talbert
Committee Member — date

Table of Contents

List of Figures

Acknowledgments

I thank many people for their help and support during my time here at AFIT. For those that I do not mention here "muito obrigada".

I thank my thesis research advisor Dr. Thomas Hartrum for giving me so much of his time, expert guidance, and perpetual patience.

I thank Major Michael Talbert for his amazing enthusiasm, for his many helpful suggestions, and for serving on my thesis committee.

I also thank Dr. Henry Potoczny for teaching me how to enjoy learning, for his unique sense of humor, and for serving on my thesis committee.

I must thank my beautiful young son Brenno for helping me to correctly pronounce my English, for always making me laugh, and for his understanding during the many times that I could not give him my full attention. His love and affection kept me focused on what is really important.

Finally, I thank my parents. I thank my father Gilberto for his ceaseless love and support. I also thank my mother Arlete for staying with Brenno and me here in Ohio, and for her constant love and encouragement. She helped me more than I can say.

Abstract

The use of many different simulation systems by the United States Department of Defense has resulted in many different scenario data representations contained in heterogeneous databases. These heterogeneous databases all represent the same data concept, but have different semantics due to intrinsic variations among the data models. In this research, I describe a unified scenario database to allow interoperability and reuse of the scenario data components while avoiding the problems of data redundancy. Using the object-oriented approach, the data and schema of the scenario databases, represented in an object-oriented model, are integrated into a global database also represented in an object-oriented model. The global database schema is extended to allow semantic interoperability of database components by explicitly associating the semantics of the schema elements of the database components with the global metadata. I create the Integration Dictionary to represent the semantic interoperability and to store the translation mappings between each database component and the global database. The Integration Dictionary also provides support for object-oriented views generation. Next, I describe a methodology to integrate databases using the Integration Dictionary. My methodology is based on an analysis of the semantics of conceptual schema elements and on the identification of related elements in the global schema. My methodology defines the resolution for schema conflicts, and associates these conflict resolutions with data changes in the Integration Dictionary. Selected parts of the Extended Air Defense Simulation (EADSIM) and Suppressor scenario databases are integrated into the global database to validate my methodology. I use the Object-Store database to implement the global database. This methodology can be applied to other systems that require database integration.

1. Introduction

1.1. Background

The Air Force Research Laboratory uses several different systems for the modeling and simulation of avionics systems. Some of these systems use Scenarios constructed by tools which require substantial user interaction and interpretation. Figure 1 provides a generic representation of the Scenario development process for the Suppressor Composite Mission Simulation System (SMCS). The current method for the development, construction and correlation of specific Scenarios is both complex and time consuming. It requires a substantial amount of manual data generation and verification [7].

When the input for the Suppressor simulation system models a military Scenario (represented as a database), the operator uses data from:

- the Multi-Spectral Force Deployment (MSFD), a flat file system containing player/system locations, identifications and a command hierarchy;
- the Digitized Terrain Elevation Data (DTED), a flat file system containing the terrain representation for the area of interest in the scenario; and
- various documents and reports that give specific threat parameters (Electronic Warfare Identification Registry - EWIR), and tactics and doctrine (Concept of Operations - CONOPS) to ensure an accurate representation of the players/subsystems in the scenario.

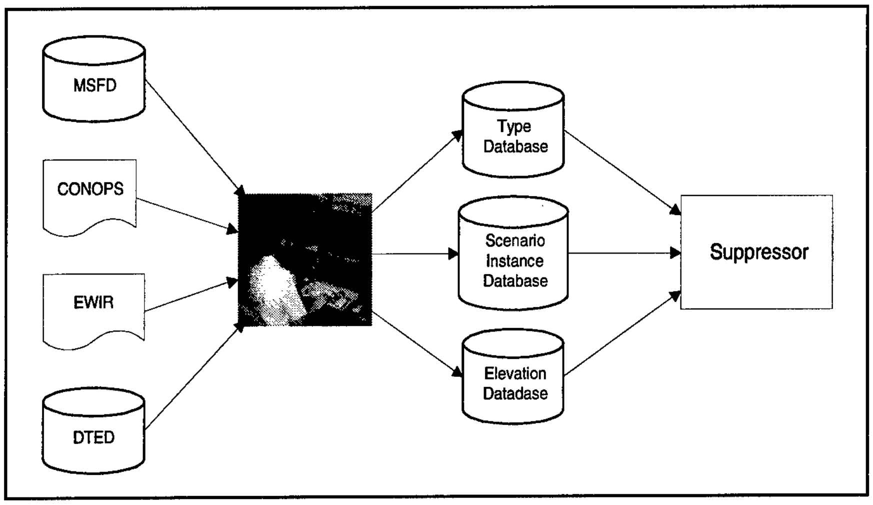


Figure 1: Scenario Creation Process for the Suppressor Simulation System

1.2. Definitions

The following terms are used in this thesis:

Scenario Development Process – The User's interpretation of input data to generate a Scenario Instance.

Model – A Class that represents an object type. The model may be part of an inheritance hierarchy.

Player – A System instance which can be part of a Scenario, e.g. an airplane, tank, building, etc.

Scenario Instance Database – A Scenario Instance is a data repository that contains an instance of a battle with specific Player information. It is normally generated from information stored in a Scenario Database. It is also called Scenario.

Scenario Database – Data repository that contains necessary information to generate a specific Scenario.

System – A set of Models. The system may contain aggregations of other Systems.

1.3. Problem

The first problem is that the current process for Scenario development in use by the Air Force Research Laboratory is very expensive. It is also very time consuming to generate new Scenario Instances because the current generation tools are uniquely designed for each individual model. Each model requires different methods for the generation of a representative database. Each model also relies on a subjective analysis by the operator for the interpretation of the EWIR and CONOPS data. This often results in discrepancies between the models, particularly in describing how they interact.

The data are similar in content for each system but they may vary due to the operator's interpretation. This reduces the correlation between systems. Additionally, differences in the use of the models prevent an automatic interpretation of the correlation.

One possible solution for this problem (Figure 2) is described below.

- Create an object-oriented Scenario Database to store all data necessary to generate a Scenario Instance and a translator program to translate the data from the object-oriented Scenario Database to the specific data model of the Scenario file used by the simulation system. This solution was addressed in Weber's Master Thesis [32].
- Create a new environment that supports an automated input interface mechanism in addition to unique input interface requirements to generate a Scenario. The Scenario Instance is generated from information stored into the object-oriented Scenario Database. The input interface was addressed in Stratton's Thesis [29].

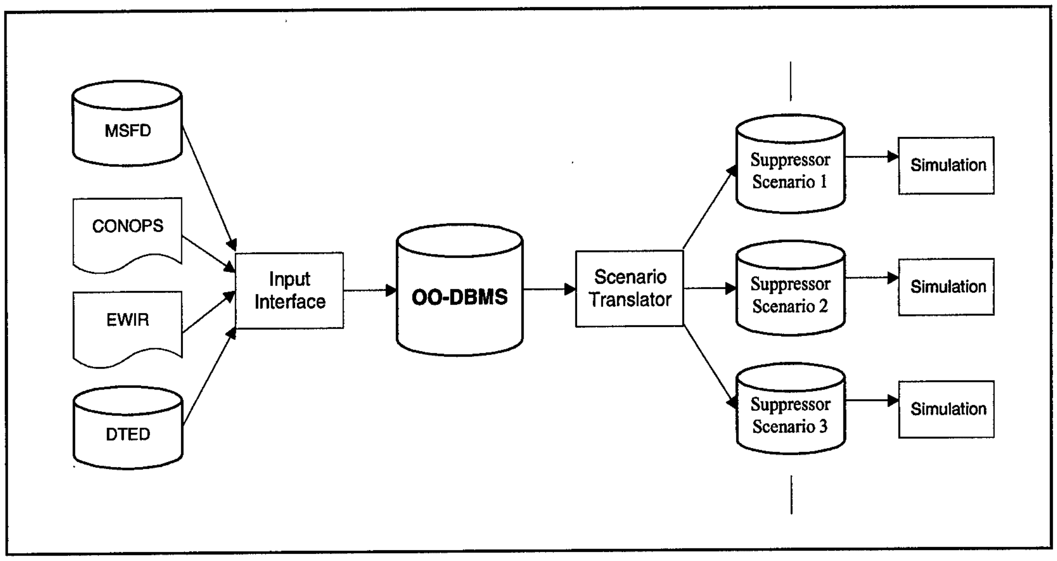


Figure 2: Object-Oriented Database for the Suppressor Simulation System

The second problem is that the integration of different simulation system scenarios is necessary to avoid data redundancy based on the assumption that simulation systems use the same data concepts. Using the object-oriented database management system (OO-DBMS) solution described above implemented by Stratton and Weber, an extension is proposed to integrate the data and the classes structure of OO_DBMSs, extracted from any simulation system scenario file into a global OO_DBMS (Figure 3).

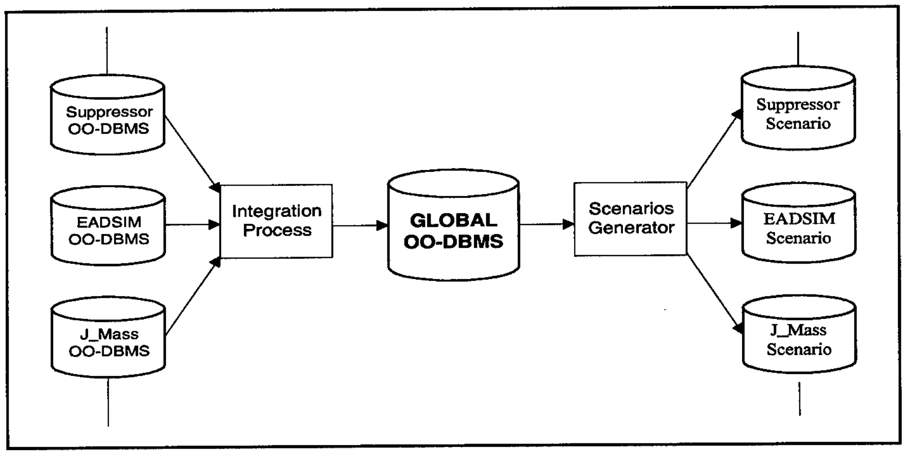


Figure 3: Global Object-Oriented Database for Simulation Systems

This solution assumes that the scenario's input data from these simulation systems were previously translated into OO_DBMSs by their respective specific input interfaces (when necessary) and the translation programs were implemented.

1.4. Problem Statement

Provide common persistent data storage to support the requirements for Scenario generation using object-oriented technology and provide methods:

- to analyze and to identify the data structure correlation and similarity between each Scenario OO-DBMS and the common data storage;
- to integrate Scenario OO-DBMSs into the common data storage; and
- to support the generation of different Scenario Databases represented in an object-oriented Model, based on information stored in the common data storage.

1.5. Scope

This research is concerned with the development of a common database structure and related data persistence using the object-oriented approach, in order to support the integration of Scenario OO-DBMSs of battle simulation systems. It is also concerned with the necessary support to generate specific Scenarios from this common data storage.

This research provides a methodology to allow the user to extend the common database to include a large variety of battle simulations' Scenario Databases.

This research does not include the implementation of user interfaces, Scenario' s object-oriented Model extraction for the new environment (OO_DBMS), since these topics were already addressed by Stratton and Weber Master's Thesis.

It also does not include the *automatic* generation of specific Scenario Databases to be used by the simulation system components of the common database.

1.6. Assumptions

In order to conduct this research, it is necessary to make a few assumptions. First, it is assumed that the existing database schemas and input structures are available. It is assumed that the sponsor will provide valid and up-to-date documents about the system and database schemas. This is necessary to complete the analysis of the current system models. It is also assumed that the data for each system is similar in content.

1.7. Approach

- *Gain an understanding of the target system* – Conduct a survey of the current database schemas and other files that are used as input data for the simulation systems.
- *Literature review* – Analyze the methodology literature about the integration of different representations of data (data models) into unique object-oriented data models, with a focus on simulation systems.
- *Learn the object-oriented language (Object-Store)* – Complete the CSCE 646 and CSCE 746 courses at AFIT.
- *Create a methodology to integrate new battle models into the database* – Define a methodology to combine different battle object models into a single global database.
- *Implement the new database* – Build a prototype global database to demonstrate the new system.

- *Verification and validation of the methodology* - Add a new model into the prototype global database to validate the methodology.

The following chapters describe my approach for solving the problem of integrating different Scenario Databases. In my approach, the integration process has two steps. The first step is the translation of the source scenario file to an object-oriented model. The second step is the integration of the newly created object-oriented model into a global object-oriented database. Chapter 2 contains analysis of the Scenario Database of the EADSIM and Suppressor simulation systems. It also contains techniques for integrating databases represented in different object models. Chapter 3 introduces my methodology for integrating databases. Chapter 4 describes how I create a prototype to implement my methodology by using the Object-Store database. My methodology is verified in Chapter 5, where the Scenario Databases of EADSIM and Suppressor are integrated into the global database. Finally, the results and conclusions of my approach, as well as topics for future research are discussed in Chapter 6.

2. Literature Review

2.1. *Introduction*

This chapter summarizes the previous work in the two major research areas that are relevant to this thesis. The first goal of this thesis was to create a common database to represent Scenario elements of different simulations. To perform this task, I completed a survey of the enabling and supporting simulation technology. The second goal was to create a methodology that allows the integration of object models from other simulation applications to the current object model. To accomplish the second goal, I reviewed the literature on methodologies concerning the integration of database schemas written in different data model representations and on the translation of battle simulation data among simulation applications. The first section below provides an historical background on the evolution of concepts of Distributed Interactive Simulation (DIS). Section 2.3 discusses the history and status of the DoD Modeling and Simulation (M&S) Master Plan. The third section discusses the simulation database scenarios for EADSIM and SUPPRESSOR, and focuses on how each scenario is structured and the differences between them. Finally, the last section presents background information on the integration of data from different data models.

2.2. *Distributed Interactive Simulation (DIS)*

To date, many researchers have focused on developing war game simulations which have a particular strength for the particular research goal. New simulations with new abilities are created sequentially, and the existing simulations are upgraded to be more accurate and realistic. One feature that many simulations have in common is standardization through the network for sharing data. This standardization has been accomplished using DIS. DIS was started in 1989 when the Advanced Research Projects Agency (ARPA) initiated a program to enhance the Simulation Network (SIMNET) program. SIMNET was developed for building a cross-country network of interactive combat simulators [23].

2.2.1. Concept

A demonstration of approximately 250 simulators successfully showed the possibilities of distributed simulation. SIMNET technology is still used today to train U.S. Army tank teams around the country. DIS is a set of protocols that carry messages about entities and events through a network. It is an interconnected, time-coherent simulation system which creates a distributed, interactive environment. DIS simulators exchange information in formatted messages called Protocol Data Units (PDUs). These PDUs provide data for the management and control of a DIS exercise and provide data related to simulated entity states and to the types of entity interactions that take place in a DIS exercise. It is possible for geographically separated simulators to interact with each other via network communications by the DIS standard [14]. DIS is designed for linking the interactive, free play activities of people in operational exercises to represent a time and space coherent synthetic world environment. This environment is created through the real-time exchange of data units between distributed, computationally autonomous simulation applications, simulators, and between instrumented equipment interconnected through standard computer network communications. In this environment, a "central" computer does not control the simulation. Instead, local computers are responsible for sending local copies of common data representing both terrain and models (e.g. tanks, fighters, and naval vessels), and remote entities based on incoming PDU messages.

2.2.2. Objectives

The basic architectural concepts are as follow [14]:

- A central computer does not control the entire simulation exercise;
- Autonomous simulation applications are responsible for maintaining the state of one or more simulation entities;
- A standard protocol is used for communicating "ground truth" data;
- Changes in the state of an entity are communicated by the simulation applications;
- Perception of events or other entities is determined by the receiving application; and
- Dead reckoning algorithms are used to reduce communications processing.

2.3. DoD Modeling and Simulation Master Plan (M&S Master Plan)

The demanding operational requirements of simulation systems led the US Department of Defense to create the Modeling and Simulation View. This demand was reflected by:

- New and more complex missions;
- Expansion of mission space;
- Increased complexity of systems and plans;
- Increased demand for joint training; and
- Security challenges (e.g. warfare).

The M&S provides readily available, operationally valid environments for use by DoD components to train jointly, develop doctrine and tactics, formulate operational plans, and access war-fighting situations. These environments also support technology assessment, system upgrades, prototype and full-scale development, and force structuring. In order to allow maximum utility and flexibility, M&S environments are constructed from affordable, reusable components interoperating through an open system architecture. The M&S Master Plan, developed in October 1995 [9], has six objectives (Figure 4). The most important is the development of a common technical framework for M&S.

1 - Provide a common technical framework for M&S
2 - Provide timely and authoritative representations of the natural environment
3 - Provide authoritative representation of systems
4 - Provide authoritative representation of human behavior
5 - Provide an M&S infrastructure to meet developer and end-user needs
6 - Share the benefits of M&S

Figure 4: Objectives of the M&S Master Plan

2.3.1. Common technical framework for M&S

The common technical framework ensures appropriate interoperability across different simulations, reuse of simulation components (entities, applications, and systems), insertion of new technologies, and flexibility to respond to changing requirements. It defines the frameworks described below.

2.3.1.1. High-Level Architecture (HLA) [8]

The High Level Architecture (HLA) is an integrated architecture which has been developed to provide a common architecture for M&S. The purpose of the HLA is to facilitate interoperability among simulations and promote reuse of simulations and their components. It is a common framework that provides major functional elements, interfaces, and design rules pertaining, as feasible to all DoD simulation applications. Any simulation developed for particular DoD components or functional areas must conform to the HLA standards. Further, HLA is being developed as an industry/DoD standard technical architecture for constructive, live and virtual interactive simulation, which supports the DIS standard. The rationale for HLA design is based on three basic premises. The first premise is that no single, monolithic simulation can satisfy the needs of all users. The second premise is that all uses of simulations and useful ways of combining them cannot be anticipated in advance. The last premise is that future technological capability and variety of operating configurations must be accommodated. HLA describes four levels of simulation, as shown in Figure 5, where each level represents a different level of abstraction. The interoperability is performed among these levels of simulation. Suppressor, EADSIM, SWENG, and SAGE are examples of mission level simulations.

In support of HLA goals, an object model (OM) is required to describe federations (set of simulations) and individual federates (one simulation). This OM identifies the data exchanged at runtime in order to achieve federation objectives.

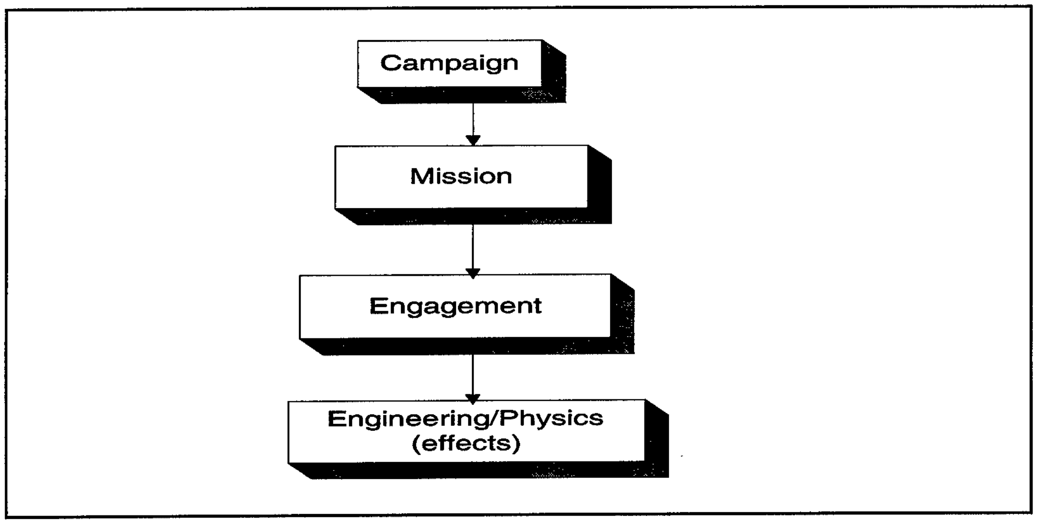


Figure 5: Levels of Abstraction in Simulation Systems

The following entities provide data support to the HLA:

- Object Model Template (OMT): a common method for recording the information contained in the required object model for each simulation. It prescribes the format and syntax for recording the information in HLA object models, to include objects, attributes, interactions, and parameters, but it does not define the specific data (e.g., vehicles, unit types) that will appear in the object models;
- Object Model Data Dictionary (OMDD): a common data framework that contains object model components consistent with DoD and other authoritative data standards;
- Object Model Library (OML): a set of commonly applicable interaction rules that allow interoperability and reuse of components through easy access to other simulation object models; and
- Automated tools for federation development, including the Data Interchanging Formats (DIF) that support the interface specifications needed for a runtime interface (RTI).

2.3.1.2. Conceptual Model of the Mission Space (CMMS)

A conceptual model is defined for each DoD mission area in order to provide a common basis (real-world view) for development of consistent and authoritative M&S representation. It is a hierarchical description of the actions and interactions among the various entities associated with a particular mission area. CMMS provides a common framework for knowledge acquisition with a standard format for expressions. This knowledge is validated and contains relevant actions and interactions organized by specific task and entity/organization.

2.3.1.3. Data Standards (DS)

DS is a common representation of data, which enables data suppliers to provide the M&S community with cost-effective, timely, and certified data to promote:

- the reuse and sharing of data;
- the interoperability of models and simulations within themselves and with the war-fighter's command, control, communications, computers, and intelligence (C4I); and
- the credibility of the modeling and simulation results.

2.4. Extended Air Defense Simulation (EADSIM)

EADSIM is a simulation of air and missile warfare ranging from few-vs-few to many-vs-many. Each platform is individually modeled and the interaction among the platforms is also individually modeled. EADSIM also models the Command and Control (C2) decision processes and the communications among the platforms on a message-by message basis. Intelligence gathering is explicitly modeled as is the intelligence information used in both offensive and defensive operations. The general areas modeled are: air defense, offensive air operations, multi-stage ballistic missiles, air breathers, sensors, jammers, satellites, early warning generic noncombatants, communications, electronic warfare, terrain, weaponry, and areas of interest.

2.4.1. EADSIM architecture

The EADSIM Scenario model is based on several processes, which are grouped into three basic functions: simulation setup, execution of scenarios, and post-processing and analysis. User interface tools are used for simulation setup and post-processing and analysis. Run-time models perform the execution of Scenarios.

2.4.2. Scenario file organization

A Scenario is made by the Scenario Generation tool during the simulation setup process. The Scenario's data is organized in a hierarchical structure in which each additional level of the hierarchy is based on the preceding lower levels. The lowest level in the hierarchy is a non-decomposable component element. Combinations of these elements are used to compound system elements, which are deployed to form platforms. Groupings of platforms are built into laydowns. The platforms are interconnected with networks, which use the protocol elements. This platform level is composed of entities, which can be defined as the unit of communication in the network. All processes in EADSIM rely on data files for storage and retrieval of definitions of everything in the scenario. The data files reflect the level of abstraction used in a Scenario. Scenarios are further combinations of lower level data. Figure 6 shows the levels of abstraction of an EADSIM Scenario.

A system is a collection of element files, which represent communication devices, jammers, sensors, weapons, radar cross sections, probability of kill, infrared signatures, wingman formation, and rule sets (guidelines to use elements). In the case of aircraft platforms, parameters that affect how the aircraft will fly can be specified.

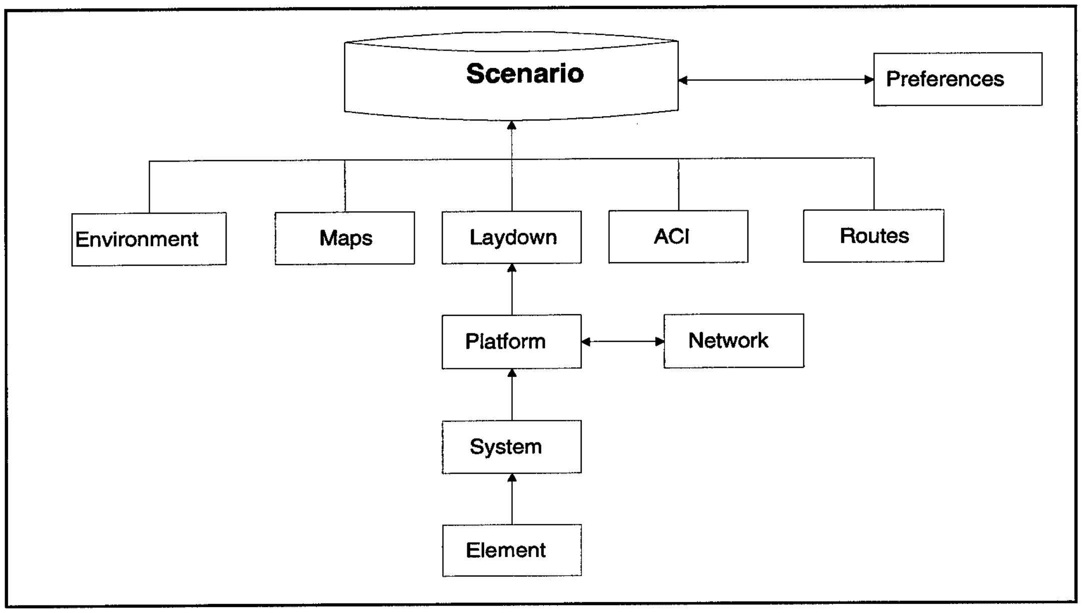


Figure 6: EADSIM Scenario Composition

2.5. *Suppressor*

Suppressor is a mission-level digital simulation that models multiple-sided conflicts involving air, ground and naval systems. The first drawback of this simulation application is the non-existence of a logic for specifying types of systems or systems interaction. The second is the non-persistence of tactics, rules of engagement, and control structure, which have to be defined as Suppressor input each time a simulation runs. The Suppressor input is based on five on-line and two off-line processes:

- Environment database (EDB) and Defense Mapping Agency (DMA): off-line processes that describe the terrain to be used with a scenario;
- Language definition: defines the input syntax definition and is executed only once, for installation;
- Type database (TDB): defines data associated with types of players;
- Scenario database (SDB): defines information specific to each occurrence of each player, such as location, movement path, zones, etc;
- Model Simulation Run (MOD): defines data to be saved for analysis and the instructions for executing the scenario; and
- Analysis database (ADB): defines data to be summarized from the run, what data should be filtered, and output statistics.

2.5.1. Scenario file organization

A player is the fundamental unit in a Suppressor Simulation. The power and flexibility in the model are derived largely from the flexibility in defining each player, its composition, and its tactics for interacting with other players. A player belongs to one or more command chains. A command chain belongs to one side. Figure 7 shows the Suppressor scenario organization.

A player is composed of one or more locations. Laydowns are used to define locations for each occurrence of a player. A location is composed of one or more elements, which are defined with susceptibility. Elements are composed of systems, which may expand resources and are defined with capabilities. Figure 8 shows the player structure.

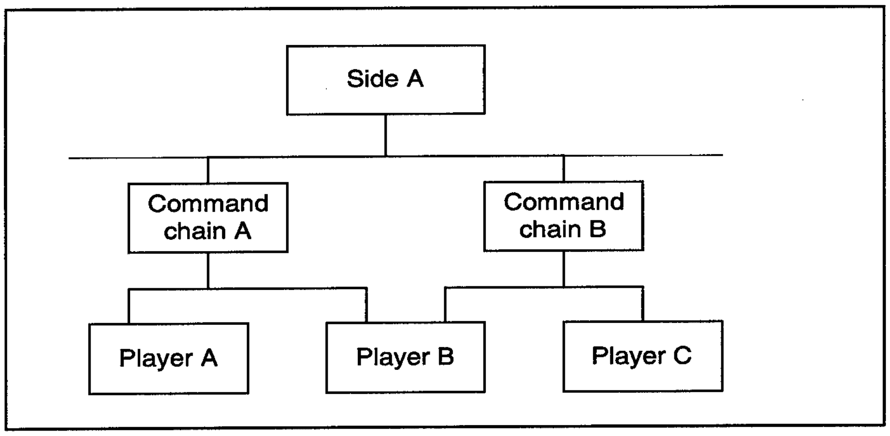


Figure 7: Suppressor Scenario Organization

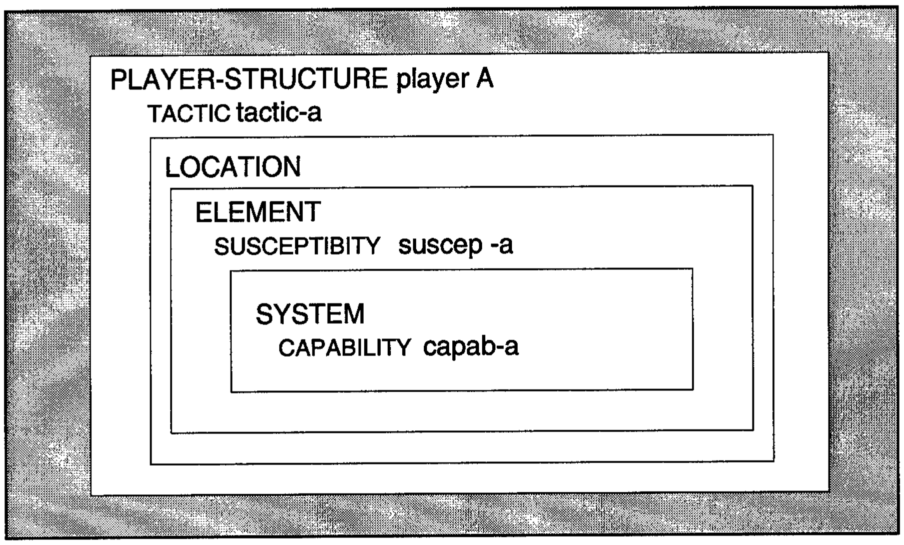


Figure 8: Player Structure

2.6. Schema integration of multi-database systems

A multi-database system (MDBS) is a confederation of autonomous and possibly heterogeneous database systems. The database systems that participate in the confederation are called local or component database systems. Creating a MDBS is a hard task since it has to deal with the heterogeneity of its component systems. This heterogeneity is caused by differences at the database system levels, including discrepancies among data models. When a component database system participates in a multi-database system, its data model is mapped to a global data model which is common to all participating systems. This global model is called a Common Data Model (CDM). The CDM allows users to access the MDBS. The conceptual schema of each component system is called a local schema. This schema is expressed in the native data model in which the system was designed. Thus, the local schemas of different component databases may be expressed in different data models.

To facilitate access to the MDBS, the translation of the local schemas into a CDM is performed. Each database participates in the federation by exporting a part of its component schema (the common part), called an export or sharable schema. A federate or global schema is created by the integration of the export schemas of the component databases.

The integration must be based on some underlying data model. The relevant integration for this thesis is object data model integration. Recently, many researchers have suggested the use of object-oriented models to build multi-database systems [3, 11, 22]. The information stored on various systems is modeled as objects, while the services of translation and access are modeled as methods of these objects. The object technology offers an efficient method for the modeling and implementation of CDM, and facilitates the use of semantic information. It generates metaclasses (classes that contain information about classes), which accommodate both heterogeneity and autonomy requirements of a MDBS. Support for heterogeneity is provided by the interface of the objects and the communication among them is independent of their internal implementation. The autonomy of the systems is respected because they may operate independently.

2.6.1. Integration of objects [22]

The schema translation solves the problem of using different data models. If during the integration process, the concepts expressed in each component schema are different, then the federate schema will be the union of all the component schemas. However, the same concepts may be expressed in different databases and furthermore, these concepts may be represented differently (heterogeneity of concepts). Users and designers must define the portion of the component schemas (sharable schema) that are to be merged.

Conflicts can occur between component schemas. The following types of conflicts are independent of the type of Common Data Model used.

- *Identity conflicts* occur when different objects in different component databases represent the same concept.
- *Schema conflicts* occur when the component schemas that represent the same concepts are not identical. This kind of conflict can be divided in two parts. *Naming conflicts* occur when the same name is used for different concepts (homonyms), or when the same concept is described by different names (synonyms). *Structural conflicts* occur when the same concept is represented by different constructs of data or when the same concept is modeled by the same construct, but the constructs used have either different structure (different or missing entities and attributes) or different behavior (different or missing methods).

- *Semantic conflicts* occur when the same concept is interpreted differently in different component databases. This category includes scale differences.
- *Data conflicts* occur when the data values of the same concept are different in different component databases.

In order to perform integration, it is necessary to identify the set of common concepts and the set of different concepts in different schemas that are mutually related by some semantic property. By applying the following outlines, these conflicts can be solved:

- *Identity conflicts* are handled by specifying which objects are equivalent. These objects could share the same object identifier (OID) or functions could be defined so that equivalent objects can be treated as the same object.
- *Naming conflicts* are handled by defining renaming operators.
- *Structural conflicts* correspond to restructuring of the class hierarchy or to modifications of the aggregate relations.
- *Semantic and data conflicts* are resolved by defining appropriate functions to the related classes in the global schema.

2.6.2. Interoperability of objects

Interoperability of multiple database systems is the capability of exchanging and combining data while, at the same time, preserving the autonomy of each existing system. Various architectures have been proposed to address the database interoperability problem [6, 11, 13, 15, 19]. Those architectures range from the integration of objects into a centralized global database to the sharing of objects among database systems while retaining their autonomy. This thesis is concerned with the identification and resolution of the semantic heterogeneity that exists between related information in different databases.

Semantic interoperability ensures that the data exchange makes sense. The user and the system provider have a common understanding of the "meaning" of the requested services and data. Generally, the semantics of procedures and data are explicit. Therefore, semantic agreements are necessary to associate semantics with data and procedure names, type definitions and type hierarchies, screen layouts and report formats. Making semantics explicit in metadata (conceptual schema) would allow designers and developers to detect mismatched assumptions and to create required mappings to overcome them.

2.6.3. Schema transformation

Transformation is a mapping from the source data model to the target data model. The first step of a transformation is to combine all original models into a combined model. During the combined model generation, a set of constraints that express interrelated data among component databases is defined. After that, the model is aligned with the scope and purpose of the global schema. The schema transformation must preserve the semantics of the original model. This alignment corresponds to the restructuring and optimizing of the combined model. The following steps are proposed for schema transformation [31]:

- Combination: integrates schemas and merges the concepts that do not present any conflict.

- Schema Restructuring: solves redundancies and conflicts that cannot be directly integrated. The transformations must be applied in conformity with the semantic constraints so that merging of classes and attributes becomes possible.
- Optimization: reduces the size of the new schema and redundancies. Primitive transformations (including merging of restructured concepts) support the schema optimization.
- View definitions: generates new mappings and translators between the global schema and the schemas of the component databases in order to access the new integrated concepts.

2.6.4. Object-model transformation

An object-model transformation is a process that changes an object-model described in some syntax into a semantically similar object-model described in a different syntax. This section presents two lists of primitive transformations on the object model. The object model characterizes the static structure and the dynamic behavior of things (objects and their behavior). The structure and behavior are regarded as a group of analogous objects (classes) in relation to their similarities and differences (generalization and specialization), and their relationships with one another (associations).

The transformation has three categories related to information content: equivalence transformation, reducing transformation, and augmenting transformation.

A primitive transformation is one that cannot be decomposed into lesser transformations. This first list [5] does not present a transformation on methods. The primitive transformations are:

1 - Transformation on a single construct type (attributes, classes or associations)

The following outline is applied to this kind of transformation:

- Remove or add constructs
- Assert a construct is not derived or derived
- Transform a multi-valued attribute
- Reorder attributes
- Transform an enumeration attribute
- Partition a construct or merge constructs
- Compose associations

2 - Transformation on multiple construct types (among more than one construct)

The following outline is applied to this kind of transformation:

- Combine associated classes or partition class
- Move an attribute across an association
- Move an attribute or association across generalization levels

3 - Transformation on a hierarchy chain

The following outline is applied to this kind of transformation:

- Remove or add a subclass
- Push sub-class up or specialize
- Fragment multiple inheritance
- Factor multiple inheritance

4 – Conversions

The following outline is applied to this kind of transformation:

- Convert generalization to/from exclusive-or associations
- Convert generalization to/from exclusive associations
- Convert link attribute to/from object attribute
- Convert association to/from class

According to [1], the primitive transformations are:

1 - Modifying the class definition

The following outline is applied to this kind of transformation:

- Modifying attributes: adding, deleting, renaming, modifying the domain, modifying the inheritance, modifying shared attributes, and transforming composite attributes into non-composite attributes.
- Modifying methods: adding, deleting, renaming, implementation, and modifying inheritance.

2 - Modifying the hierarchy chain

The following outline is applied to this kind of transformation:

- Making super-class
- Removing super-class
- Modifying order of super-classes

3 - Modifying the set of classes

The following outlines is applied to this kind of transformation:

- Creating classes
- Deleting classes
- Renaming classes

2.6.5. The Remote-Exchange Experimental Prototype System [11]

This architecture extends the traditional federated architecture in supporting two key aspects: first, to discover the location and content of related non-local information units (simple objects, types of objects, units of behavior, etc.); second, to identify and resolve the semantic heterogeneity that exists between related information in different database components. In order to achieve interoperability, a common model for describing the sharable data must be defined and utilized. This model should be semantically expressive enough to capture the meanings of conceptual schemas, which may reflect heterogeneity. It supports the following basic features of object-oriented models:

- Complex objects (aggregation);
- Type membership (classification);
- Sub-type to super-type relationships (generalization); and
- Inheritance of properties (attributes) and user-defined functions (methods).

Figure 9 illustrates the Remote-Exchange experimental system.

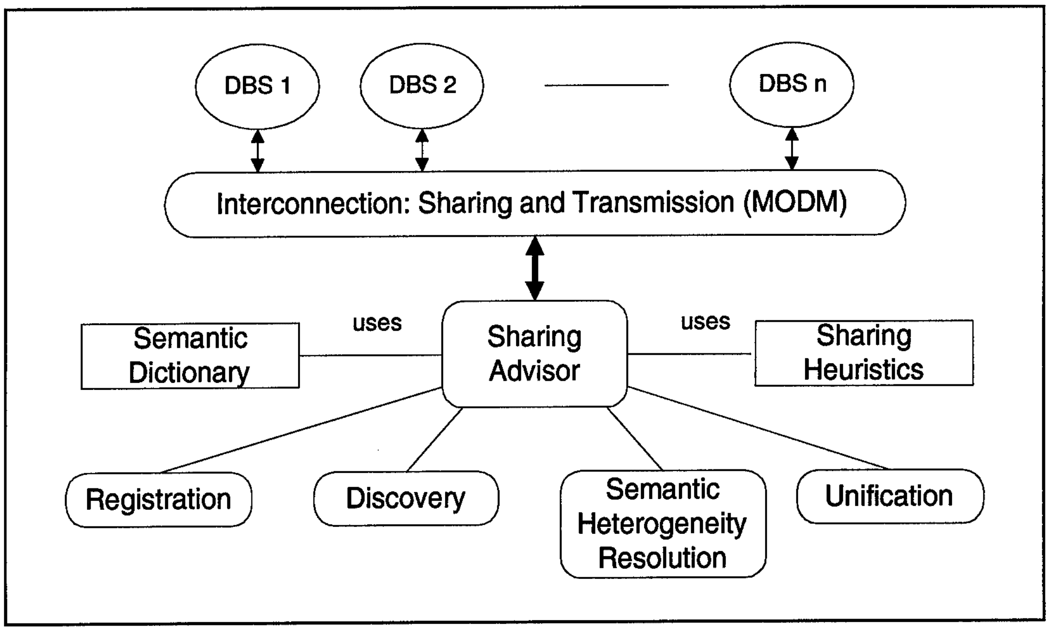


Figure 9: The Remote-Exchange architecture

The *Sharing Advisor* manages the Semantic Dictionary which holds the knowledge about existing type objects. It also provides the following services: Registration, Discovery, Semantic Heterogeneity Resolution and Unification.

The *Semantic Dictionary* is based on a dynamic concept hierarchy. Its increasing size depends on the knowledge and information received from the sharing advisor. A type object represents a specific view of a corresponding real world concept, and it is tailored to the focus and interest of the related database. Therefore, the set of properties associated with the type object can be viewed as a subset of those associated with the real world concept.

Sharing Heuristics provides the distinguishing capability of a property with respect to a concept. This allows the sharing advisor to determine if the meaning of a type object that is being registered can be determined based upon its properties, or whether further assistance from users is necessary.

The *Registration* process allows a new system to integrate its schema into the shared common schema. It establishes the initial sharing context within the federation by logically connecting the exported information to the Semantic Dictionary via the sharing advisor. Incremental registration allows a system to augment the common schema with new information. The newly acquired knowledge and the newly registered information are stored in the Semantic Dictionary.

The *Discovery* process identifies appropriate information relevant to a request of a component system that is initializing a sharing procedure. This operation is based on three kinds of discovery requests between the common schema and the exported schema of the new system to be integrated: similar concepts, complementary information, and overlapping information.

The *Semantic Heterogeneity Resolution* process determines how discovered information can be unified with local data (conceptual schema) of a system component due to semantic discrepancies that may exist between related concepts in different systems.

The *Unification* process allows the non-local object(s) to be unified with the corresponding local object(s). It happens after the semantic heterogeneity resolution is performed. In some cases, the local metadata framework must be restructured to achieve a result that is complete, minimal, and understandable.

2.7. Summary

In Chapter 1, I propose the use of a unified Scenario Database to integrate different Scenario Databases. Since the simulation systems must be autonomous, I choose the object-oriented technique to implement the common database. The objects stored in the common database respect the heterogeneity and autonomy of the source implementation. In this chapter, I present the techniques for building a common unified object-oriented database. The techniques are used as a base to construct a unified Scenario Database. I present a unified analysis of the process of building an object-oriented database from heterogeneous databases. Also, I show a framework for identifying conflicts during the integration of component databases into the unified database. The definition of the schema conflicts in my methodology is based on this framework. I introduce the Integration Dictionary to support the registration of similarity among type objects, which is critical to the success of this thesis.

I load part of the EADSIM Scenario Database into the common database in order to serve as a base for data integration. I use part of the Suppressor Scenario to validate the methodology for integrating databases. I describe the methodology next in Chapter 3.

3. Methodology for integrating databases

3.1. Introduction

Users of a shared database usually need to access just part of the stored data, which relates to their own application systems. Views provide the data independence necessary to implement this kind of data access. Several researchers have investigated the integration of heterogeneous data repositories via object-oriented views [2, 3, 28], since they can provide interoperability by hiding the idiosyncrasies of the component systems' schemas to be integrated into one unified, federated system schema. In Ra's approach [25], all objects are associated with a single underlying global schema and each version of the schema is implemented via a view defined on the global schema. In Hammer's approach [11], objects from different sources are integrated into a single global schema, which uses an object-based model to implement the interoperability. Hammer also proposes a system for automatically identifying and accessing these non-local objects.

I propose the use of a unified object-oriented global schema to implement the integration of battle simulation scenarios. This unified global schema is used as a basis for views of the different scenario schemas. It also provides a standard metamodel in order to abstract the models to be integrated and their respective connections with the metamodel. All models must be compared to the metamodel standard concept (represented by the global schema), thus providing integration through a standardization related to the semantics of data. The metamodel may be incomplete and cover only those elements of models for which integration is necessary. Therefore, I created a global schema (introduced in Chapter 5) to support the exported schemas of different simulation system schemas. Based on the global schema, I create an object-oriented database to store the Scenarios' data in a common global format. Any exported schema that is to be integrated into the global schema needs to be created by the Integration Administrator (IA). The IA translates the local database schema into a common intermediate representation.

I also propose the Integration Dictionary. This dictionary registers the translation relations between local schemas and the global schema. The Integration Dictionary defines the conceptual objects, attributes, and methods, and relates them to the local objects, attributes, and methods of each simulation system, which are parts of the global schema. By using the object-oriented approach the global schema will integrate different modeling schemas into one consistent battle simulation scenario model. The access to shared data is performed through the global schema interfaces.

This methodology is based on the following ideas:

- all of the component scenario schemas are already translated into the common data model (object-oriented model);
- integration of part of the component schemas (exported part) and data into the common global database;
- specification and implementation of the Integration Dictionary, which is composed of the metadata of the unified global schema; and
- specification of the User's Views (specific Scenario Databases) of the common global database to be used by the scenario generators of the simulation system components. The User's Views are generated using global schema interfaces (methods).

The remainder of this chapter is organized as follows. Section 3.2 describes the scenario's integration architecture. Section 3.3 describes the Integration Dictionary. Section 3.4 describes the conflict resolutions to be applied when integrating different exported schemas into the global schema. Section 3.4 describes the methodology for integrating a new schema into the existing global schema. Section 3.5 describes the schema modification operations, which are applied using the conflict resolution. Finally, Section 3.6 defines the methodology of integration as ordered steps that must be followed and implemented when necessary.

3.2. The scenario's integration architecture

Integration of dissimilar scenario databases requires a pre-translation of their schemas and data into a common intermediate representation. Then, the schemas are integrated. The proposed integration architecture (Figure 10) includes a model that integrates source scenario files of different simulation systems into a unified global object-oriented database. In order to integrate a new source scenario file into the global database, the source file must first be translated into an intermediate representation in the object-oriented model (OO-Model), called the exported schema. This OO-Model is implemented in an object-oriented database, called the User's View. Second, the schema and data of the User's View are integrated into the global database. In contrast, to generate a specific simulation system scenario, the data stored in the global object-oriented database must first be accessed and transformed back into the specific simulation system's intermediate representation in the OO-Model. Then, the specific OO-Model representation must be translated back into the source scenario file. The translation between the OO-Model and the source scenario file are beyond the scope of this research.

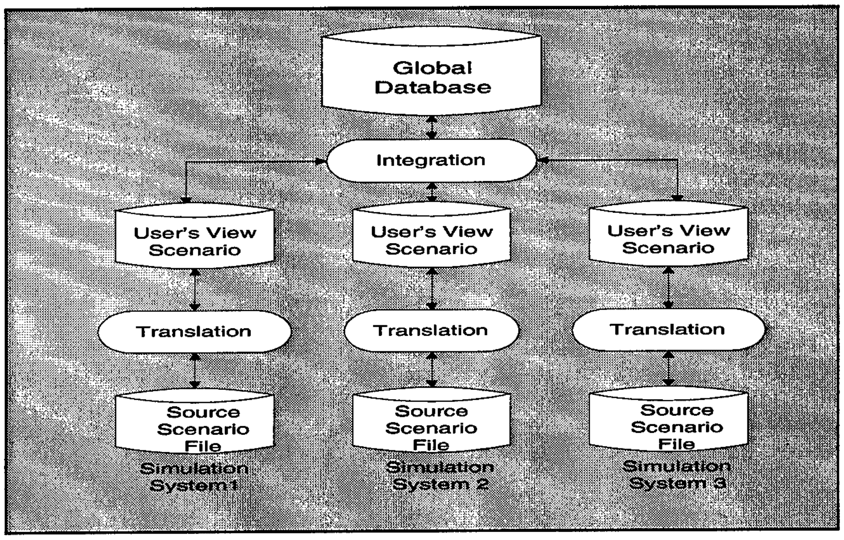


Figure 10: Scenario's Integration Architecture

3.2.1. Generation of a specific object-oriented model (exported schema)

Each local database schema can be defined using different data models such as a file system, relational model, entity-relationship model, or an object-oriented model. The schema translation is performed when a schema represented in one data model is mapped to an equivalent schema represented in a different data model. Therefore, any schema that is not represented in the object-oriented model must be translated. This translation is the subject of several research efforts [10, 12, 17, 32]. The Integration Administrator (IA) must have the knowledge to perform re-engineering of the heterogeneous databases in order to perform the translation. Figure 11 shows an example of a schema translation from the relational model to the object-oriented model.

Relational Database	**Object Model**
Table	Class
Row	Object Instance
Column	Attribute
Primary key. A table to store the Primary Key, the Table Name and the Object ID must be created. The Primary Key would be the Object Key. With this approach, the facility of RDBMS in terms of navigation and referential integrity is kept.	Object Key and Object ID (OID)
Dependent table. The Primary Key of the parent table is a Foreign Key in the dependent table. The Primary Key of the dependent table forms the Object Key and the Primary key of the parent table is not mapped.	Relationship "is-a". The inheritance definition must be inserted for this table.
Relationship "1:n" and "n:1". The Primary key of the "1" side table is a Foreign Key in the "n" side table. Foreign key is not mapped.	Multi-valued attribute is created in the 1 side table to reference the n side table objects (single attribute is created in the n side table to reference the 1 side table object)
Relationship "1:1". The Primary key of either table is the Foreign Key in the other table. Foreign key is not mapped.	Single attribute is created in the one of the table to reference the other side table object (or in both tables)
A table in a relational database represents an aggregated object. So, the Primary Key of the table, which has the aggregated object, is the foreign key in the table that represents the aggregate. Foreign key is not mapped.	Aggregation "part-of". A single attribute in the table that contains the aggregated object is created to reference that object.
Relationship "n:n". A third table is used to represent the relationship "n:n". The Primary key of this third table is formed by Foreign Keys, which are the Primary Keys of the tables in the relationship. Foreign keys are not mapped.	Multi-valued attribute is created in one of the n side table to reference the other n side table objects (or created in both tables)
Stored Procedures* and procedures to keep referential integrity	Methods

* Procedures that are activated on insertion, update and deletion of attributes or tables.

Figure 11: Example Transformation from a Relational Model to an Object-Oriented Model.

This operation generates the mappings that correlate the original schema to the translated schema, now represented as an object-oriented model. The translation mappings of each simulation system scenario file must be stored in order to be applied in further translations (from an OO_Model to a source model) of new scenarios. The translation operation and mappings is not in the scope of this research. An example of the translation operation and mappings (for Suppressor and SWEG) can be found in Weber's research [32].

3.2.2. Integration of specific object-oriented model

The Integration Administrator (IA) also performs the integration of a specific OO-Model into the global Model. This thesis provides a framework for comparing, solving the conflicts, restructuring, and merging the schemas. After the schema translation of the source scenario file is performed, the User's View as represented in an OO-Model is generated. The schema and data of the User's View must be integrated into the unified global database.

The integration operation also generates mappings for each specific User's View component, which is stored in the *Integration Dictionary* described in the next section.

The translation and integration mappings can be used for two kinds of transformations back to the original file. The first transformation is performed before running the scenario generator of the specific simulation system (Figure 12). It is performed in two steps:

- Transformation of the global data (represented in an OO-Model) stored in the global database into the specific scenario User's View, also in an OO-Model; and
- Translation of the specific scenario User's View to the original scenario file.

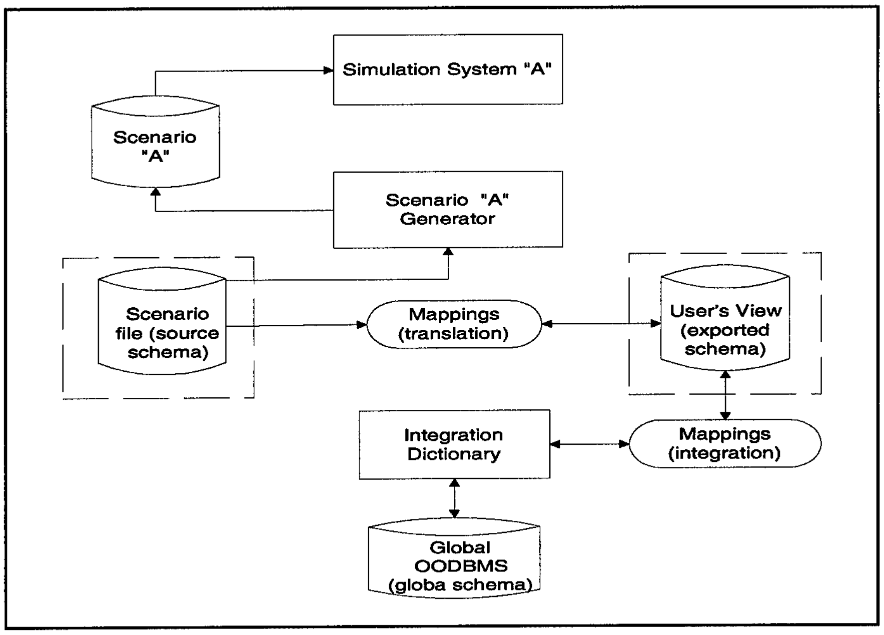


Figure 12: Generation of a Scenario

The second is a direct transformation of the specific simulation system scenario from the global object-oriented model to an original scenario file during the execution of the related scenario generator. The scenario generators could use these mappings (translation and integration) to directly access common data from the global database and generate a new scenario (second solution). The automatic translation and integration mappings are shown in Figure 13.

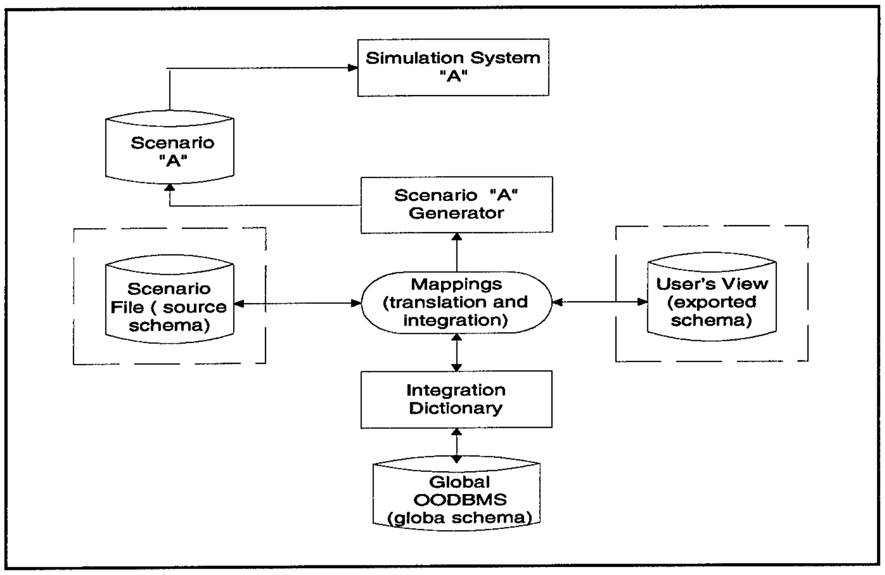

Simulation System "A"
Scenario "A"
Scenario "A" Generator
Scenario File (source schema)
Mappings (translation and integration)
User's View (exported schema)
Integration Dictionary
Global OODBMS (globa schema)

Figure 13: Generation of a Scenario

The first solution is the one adopted for this thesis, and transforms the required data from the global schema to the object-oriented scenario database, which must be further translated into the original scenario file in order to run the simulation. The focus of this thesis is to address the problem of schema integration.

The translation into a common representation reduces the number of translations as introduced in [21], which relates two schemas at a time. For each existing schema, translations to all other schema components are required and must be stored. The translations are translation of the source schema to a prototype and translation of the prototype to the specific target schema. The approach used in this thesis facilitates the integration of new simulation system Scenario Databases into the common Scenario Database since it does not use prototypes. The schema translations of a specific Scenario Database to be integrated into the global database are required just for the specific schema and the global schema. It also facilitates access to the information since the access to the common database is made by the global database interfaces.

3.3. Integration Dictionary

The common data model, created from the integration process of heterogeneous simulation system components, must be semantically expressive enough to capture the intended meanings of each exported schema. The basic object-oriented model lacks some concepts necessary for a common data model. Thus, it is necessary extend the data model to facilitate unification and integration. The Integration Dictionary supports the unification and integration of models based on the exported schemas of battle simulations, providing the required extensions. It is a metamodel that allows exported scenario schemas of each component simulation system to be mapped to a common concept while storing the semantic properties.

3.3.1. Extensions of the object-oriented common data model

The first required extension is the *extension of classes*. This extension stores the set of all object instances that "belong-to" a class. A hierarchy of extensions of classes must exist when a related class hierarchy exists. For example, if class A is a subclass of class B then the extension of class A is a subset of the "deep" extension of class B. Each time new instances are created or deleted the extent of a class changes.

The second required extension is the set of *schema evolution operations* required to represent a common data model. These operations dynamically define and modify the global database schema, such as the class definitions and the inheritance structure. These operations play an important role in restructuring the global schema resulting from the merging of component schemas. The operations are:

- Adding and deleting classes;
- Adding and deleting members of classes;
- Changing the type of a class member;
- Renaming an attribute;
- Initializing instances of a new attribute from attributes of old or existing instances;
- Moving attributes to/from a derived or base class; and
- Adding or adjusting references.

Finally, the last required extension is the *semantic extension.* This extension is necessary to capture the semantics of component schemas and their relationships with the global unified schema. They can be implemented using the metaclass mechanism of the basic model to provide a mapping between global and component schemas.

3.3.2. Integration Dictionary architecture

The Integration Dictionary is an application system that extends the OO-DBMS capability to support the unified global schema. This system is constructed on top of the global scenario database. It consists of conceptual classes and methods. It provides access to metadata defined for each conceptual class in the Integration Dictionary. Classes determined to be similar by the Integration Administrator are classified into a collection called aliases within the related conceptual class. The aliases set increases after the schema integration of a new specific Scenario Database into the global database. A conceptual class can be part of a concept hierarchy and inherits the attributes and methods of the super-conceptual class. The conceptual class is composed of class-aliases, attribute definitions, method definitions and the real class identifier as shown in Figure 14.

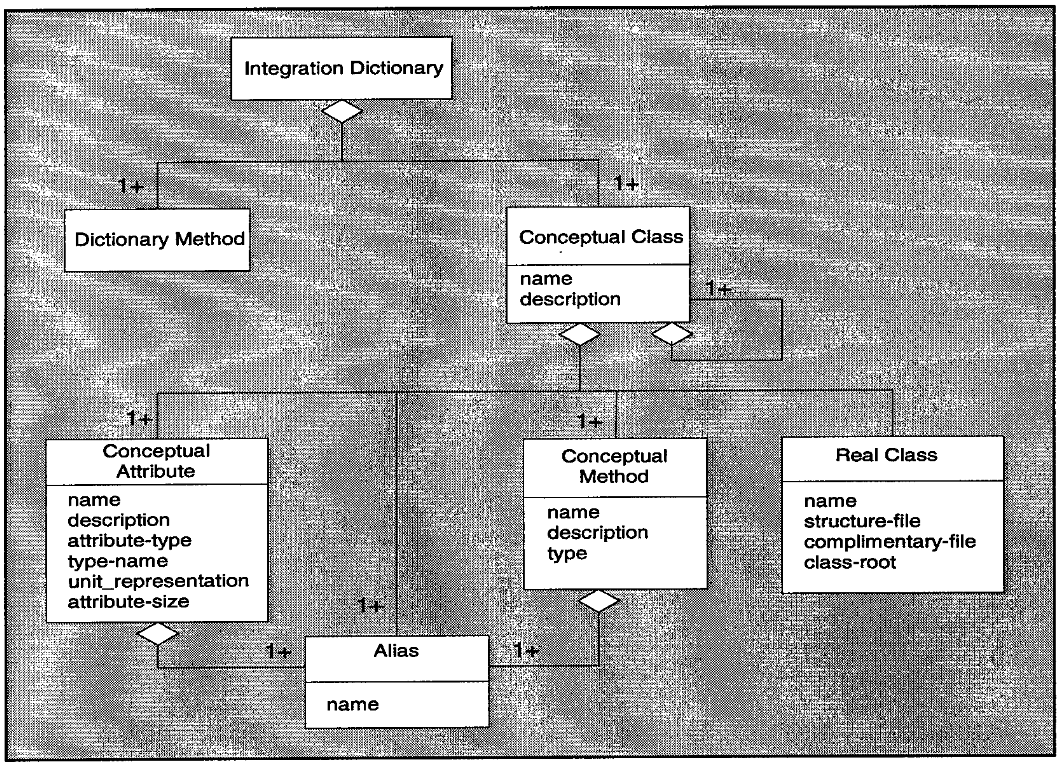


Figure 14: Integration Dictionary model

3.3.2.1. Dictionary Methods

These methods are data conversion procedures. The dictionary methods consist of the method signature and method code. The creation of the dictionary methods is necessary when inconsistencies between attribute representations or data types of semantically related attributes are detected during the schema integration process (specific schema and global schema). The implementation of dictionary methods is addressed in Section 3.4.

3.3.2.2. Conceptual Classes

Conceptual classes are created in order to represent the set of exported classes that express similar concepts. These exported classes are merged during the integration process. Conflicts between elements of both schemas that are being merged may occur. In order to solve these conflicts the conceptual class must be restructured and eventually may form a conceptual hierarchy. The Conceptual-Class is composed of:

- *Conceptual class name*: The conceptual name must reflect a high level of abstraction of a concept in order to represent entities with a similar concept.
- *Description*: Describes the concept and the class functionality.
- *Super-conceptual classes*: This aggregate component identifies the immediate parent(s) class(es) in an inheritance hierarchy.
- *Attribute definitions*: This is an aggregate component called a conceptual attribute.
- *Method definitions*: This is an aggregate component called a conceptual method.
- *Real Class definitions*: This is an aggregate component called real class.
- *Class-aliases*: This is an aggregate component called Alias. It holds exported class names similar to the related concept. This is necessary since these names were mapped from the data models of individual simulation systems into the common data model.

The sub-sections 3.3.2.3 to 3.3.2.6 describe each element of the Conceptual-Class.

3.3.2.3. Conceptual Attribute

This object describes the metadata of an attribute. It is composed of a Conceptual-Attribute name, description, attribute-type, type-name, unit of representation, attribute-size and Attribute-Aliases. Attribute-Aliases is an aggregate component called Alias, which holds all the Conceptual-Attribute names that were defined for the exported class represented by the Conceptual-Class. The attribute type holds three kinds of attributes:

- single type: represents single types or atomic types such as string, integer, etc;
- object type: represents a user-defined class; and
- multi-valued type: represents set, bag, list, collection, etc. of single types or object types.

3.3.2.4. Conceptual Method

This object describes the metadata of a method. It is composed of a conceptual method name, description, type and Method-Aliases. Type describes the function of the method. Example types include Get, Set, Constructor, and Destructor. Method-Aliases is an aggregated object called Alias, which holds all the method names that were defined for the exported class represented by the Conceptual-Class.

3.3.2.5. Alias

This object is composed of *name* and is used to represent similar names of classes, attributes and methods of component simulation systems related to Conceptual-Classes, Conceptual-Attributes and Conceptual-Methods respectively.

3.3.2.6. Real Class

This object is the Integration Dictionary representation of the actual global schema for the related conceptual class. It is composed of the Real-Class name (this would usually be the same as the Conceptual-Class name, but is subject to the naming rules of the OO-DBMS), filenames of the class structure (user defined type) represented in the OO-DBMS, and the Real-Class class-root (root of the extent of a class). The corresponding Real-Class schema in the OO-DBMS contains all the attributes defined by the Conceptual-Attributes and all the methods defined by the Conceptual-Methods. The class-root attribute holds the root name that points to the extent of a class, which contains all the instances of the class.

3.4. Conflict Resolution

Integration of schemas from independent databases will result in conflicts most of the time. The concepts implicit in classes of both databases being merged may be inconsistent. The schema conflicts described in this section are based on Kim's approach [15] where several schema conflicts are described. I selected the relevant conflicts for this work to be used in this section. In my approach, the resolutions for the schema conflicts between two schemas (the new schema to be integrated represented in an object-oriented model and the conceptual schema represented in an object-oriented model) use the Integration Dictionary. I also define the Integration Dictionary changes and global schema updates for each kind of conflict. The use of the Integration Dictionary facilitates the schema integration and avoids schema evolution for naming conflicts and some of the schema evolutions for structural conflicts. The sub-sections of this section describe the selected schema conflicts grouped by categories: name, representation, class member and schema. For each conflict the related conflict resolution, changes in the Integration Dictionary and in the global schema code, as well as necessary schema evolution operation are described.

3.4.1. Name conflict

- Class Name

Conflicts with the class name can arise when the name of the new (exported) class is not defined in the set of Class-Aliases of the similar concept class in the Integration Dictionary.

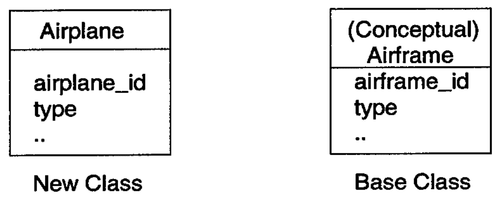
Resolution: Associate the semantically related class names.

Integration Dictionary changes: The new class name must be included in the Class-Aliases set of the Conceptual-Class (Example 1).

Global schema code changes: No schema change is necessary.

Global schema evolution operation: No schema evolution is necessary.

Example 1



Conceptual Class: AIRFRAME
Conceptual class name: airframe
Description: object designed to be capable of atmospheric flight
Super-conceptual class: null
Class-aliases: [{aircraft}]
Attribute Definition: [Conceptual attribute name: airframe_id
Description: unique identification of the airframe
Attribute-aliases: [{aircraft_id}]
Attribute-type: Single
Attribute-type-name: string
Attribute-size: 30
Unit of representation: null]
Method Definition: [Conceptual method name: get_airframeid
Description: get unique identification of the airframe
Method-aliases: [{get_aircraftid}];
Conceptual method name: set_airframeid
Description: set unique identification of the airframe
Method-aliases: [{set_aircraftid}]]
Real Class Definition: [Class name: airframe
Class structure file: airframe.hh
Class complementary file: airframe.cc
Class-root: airframe_root]

New class to be integrated: Airplane
Change: Class-aliases: {aircraft, airplane}

- Attribute Name

After identifying the concept of the new (exported) class, the new (exported) attributes are compared with the related Conceptual-Attributes. Conflicts with the attribute name will arise when the name of the new attribute is not defined in the set of Attribute-Aliases of the similar concept attribute in the Integration Dictionary.

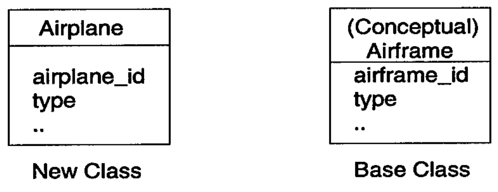
Resolution: Associate the semantically related attribute names.

Integration Dictionary changes: The new Conceptual-Attribute must be included in the Attribute-Aliases set of the Conceptual-Attribute (example 2).

Global schema code changes: No schema change is necessary.

<u>Global schema evolution operation:</u> No schema evolution is necessary.

Example 2



Conceptual Class: AIRFRAME
Conceptual class name: airframe
Description: object designed to be capable of atmospheric flight
Super-conceptual class: null
Class-aliases: [{aircraft, airplane}]
Attribute Definition: [Conceptual attribute name: airframe_id
Description: unique identification of the airframe
Attribute-aliases: [{aircraft_id}]
Attribute-type: Single
Attribute-type-name: string
Attribute-size: 30
Unit of representation: null]
Method Definition: [Conceptual method name: get_airframeid
Description: get unique identification of the airframe
Method-aliases: [{get_aircraftid}];
Conceptual method name: set_airframeid
Description: set unique identification of the airframe
Method-aliases: [{set_aircraftid}]]
Real Class Definition: [Class name: airframe
Class structure file: airframe.hh
Class complementary file: airframe.cc
Class-root: airframe_root]

New attribute to be integrated to the conceptual attribute airframe_id: airplane_id
Change: Attribute-aliases: {aircraft_id, airplane_id}

- Method name

After identifying the concept of the new (exported) class and comparing names, the new (exported) methods are compared with the related Conceptual-Methods. Conflicts with the new method name will arise when the name of the new method is not defined in the set of Method-Aliases of the similar Concept-Method in the Integration Dictionary. In this case the new method uses the same conceptual parameter(s) as defined in the related Conceptual-Method.

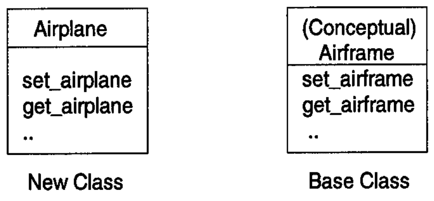
<u>Resolution:</u> Associate the semantically related method names.

Integration Dictionary changes: The new method must be included in the Method-Aliases set of the Conceptual-Method (example 3).

Global schema code changes: No schema change is necessary.

Global schema evolution: No schema evolution is necessary.

Example 3



Conceptual Class: AIRFRAME
Conceptual class name: airframe
Description: object designed to be capable of atmospheric flight
Super-conceptual class: null
Class-aliases: [{aircraft, airplane}]
Attribute Definition: [Conceptual attribute name: airframe_id
Description: unique identification of the airframe
Attribute-aliases: [{aircraft_id, airplane_id}]
Attribute-type: Single
Attribute-type-name: string
Attribute-size: 30
Unit of representation: null]
Method Definition: [Conceptual method name: get_airframeid
Description: get unique identification of the airframe
Method-aliases: [{get_aircraftid}];
Conceptual method name: set_airframeid
Description: set unique identification of the airframe
Method-aliases: [{set_aircraftid}]]
Real Class Definition: [Class name: airframe
Class structure file: airframe.hh
Class complementary file: airframe.cc
Class-root: airframe_root]

New method to be integrated to the conceptual method get_airframe: get_airplane
Changes in Method get_airframe: Method-aliases: {get_aircraft, get_airplane}
Changes in Method set_airframe: Method-aliases: {set_aircraft, set_airplane}

3.4.2. Representation conflict

- Expressions

This conflict arises when both schemas that represent the same piece of information use different expressions (scalar values).

Resolution: Define an isomorphism between the values. This can be achieved by creating a static lookup table (example 4).

Integration Dictionary changes: Add a new dictionary method in the Integration Dictionary schema (to implement the resolution described above) in addition to the creation of a new Conceptual-Method in the related Conceptual-Class.

Global schema code changes: Add the new method signature and code in the related global class schema.

Global schema evolution operation: Update the global schema structure.

Example 4

Table look-up for different codes denoting distinct flight modes:

1	BALLISTIC	B
2	FLIGHT PATH ANGLE CURVE	C
3	FLY TO POINT	P

- Units

This conflict arises when numeric data denoting the same physical quantity are represented in different units in both schemas.

Resolution: Define a method to convert the numeric value in one unit to another (example 5) and to provide this functionality to other classes that need the same kind of conversion.

Integration Dictionary changes: Add a new dictionary method in the integration dictionary schema (to implement the resolution described above) in addition to the creation of a new Conceptual-Method in the related Conceptual-Class.

Global schema code changes: Add the new method signature and code in the related global class schema.

Global schema evolution operation: Update the global schema structure.

Example 5

The height is stored in different units in database 1 (cm) and 2 (inches). The unit chosen for integration is inches. The appropriate method conversion must be created, in which height in cm is converted to inches:

```
float Convert_cm_inches (float value)
{  float v_inches = value/2.54;
   return v_inches; }
```

- Precision

This conflict arises when semantically equivalent attributes draw values from domains with different cardinalities. This difference in cardinality results in different scales of precision for similar data. In example 6, database 1 defines three possible values for velocity-category attribute: high, medium, and low. Database 2 does not have this kind of attribute, but it defines bounds of velocity that give the category of the velocity. It implies that while database 2 represent the information using two attributes, database1 represents the same information using one attribute.

Resolution: The solution is to define an isomorphism between the values. This can be achieved by creating a static lookup table to map domains of semantically equivalent attributes.

Integration Dictionary changes: Add a new dictionary method in the integration dictionary schema (to implement the resolution described above) in addition to the creation of a new Conceptual-Method in the related Conceptual-Class.

Global schema code changes: Add the new method signature and code in the related global class schema.

Global schema evolution operation: Update the global schema structure.

Example 6

Table look-up for different code representation (database 1 uses string code whereas database 2 numeric code in knots) of velocity category:

Velocity category	Lbound Velocity	Ubound Velocity
HIGH	900	
MEDIUM	300	899
LOW	0	299

3.4.3. Class member conflict (between new class and semantically equivalent conceptual class)

- Attribute composition

This conflict arises when there are structural differences in both classes such that the domain of a semantically equivalent attribute in the Conceptual-Class is a user-defined class, whereas that in the new (exported) class is an atomic type.

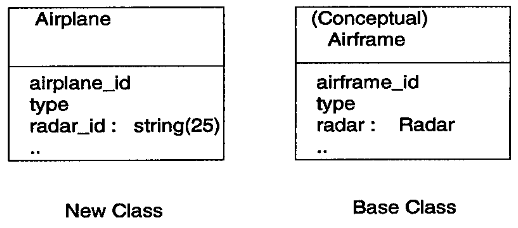
Resolution: The domain must be homogenized. A new method must be created in order to transform the domain projection (example 7).

Integration Dictionary changes: A new Conceptual-Method must be created in the related Conceptual-Class.

Global schema code changes: Add the new method signature and code in the related global class schema. The new method must project the atomic type instead of the whole class.

Global schema evolution operation: Update the global schema structure.

Example 7



The domain of attribute radar_id in Airplane is string whereas in the Base (conceptual) Class Airframe is the (conceptual) Class Radar. Therefore, a conceptual-Method to project the radar_id used by the new class must be created in the Conceptual-Class Airframe

- Attribute data type

This conflict arises when the domains (types) are different in both schemas for semantically equivalent attributes.

Resolution: Convert the domains. If automatic conversion is not defined in the database language that implements the Integration Dictionary, the IA must supply an explicit conversion (example 8).

Integration Dictionary changes: A dictionary method must be created in order to provide this functionality for all semantically related attributes that need the conversion, besides the creation of a new Conceptual-Method in the Conceptual-Class.

Global schema code changes: Add the new method signature and code in the related global class schema.

The new method in the global class schema calls the related conversion dictionary method defined in the Integration Dictionary.

Global schema evolution operation: Update the global schema structure.

Example 8

The Attribute A in database 1 is float and in database 2 is integer. The unit chosen for integration is float. The appropriate method conversion must be created in order convert to integer to float:

```
float Convert_int_float (value: int)
  return itof(value);
```

- Attribute concatenation

This conflict arises when information captured by a single attribute in the new (exported) class is equivalent to that in more than one attribute belonging to the related Conceptual-Class.

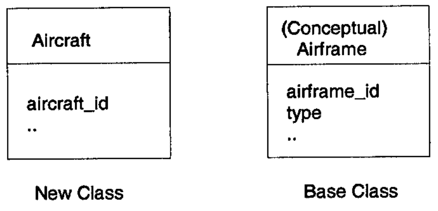
Resolution: A method must be defined to concatenate attributes (example 9).

Integration Dictionary changes: A new Conceptual-Method must be created in the related Conceptual-Class.

Global schema code changes: Add the new method signature and code in the related global class schema.

Global schema evolution operation: Update the global schema structure.

Example 9



The Attribute aircraft_id in New Class is composed of aircraft type and the aircraft_id, and in the Base (conceptual) Class Airframe these two attributes are distinct. Therefore, a method to created the concatenation of type + airframe_id must be created.

- Missing attribute

This conflict arises when the number of attributes in the new (exported) class is greater than in the related global class.

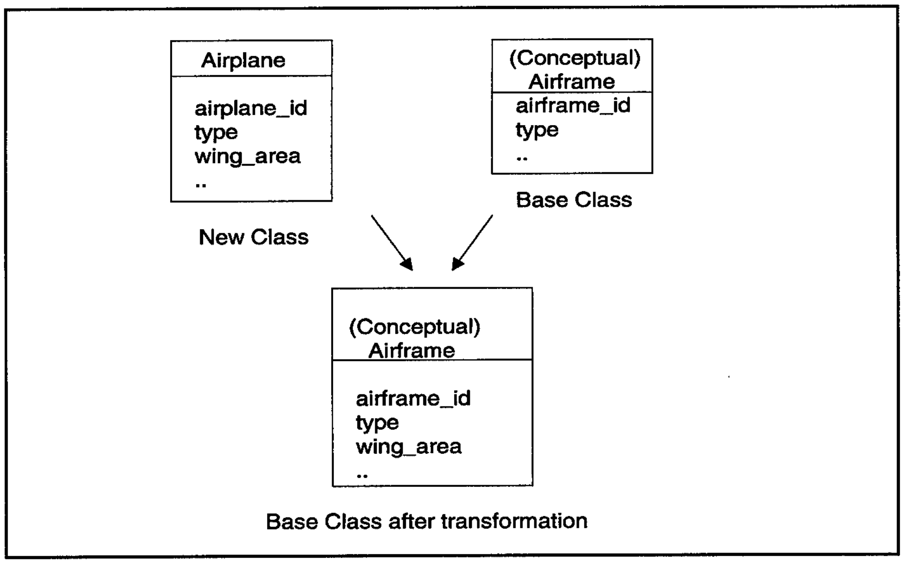
Resolution: Insert the new attribute in the related global class.

Integration Dictionary changes: The Conceptual-Attribute must be created in the related Conceptual-Class in the Integration Dictionary (example 10). The conflict resolution for attribute names must be applied and all required data must be inserted into the newly created instance.

Global schema code changes: Add a new attribute in the related global class schema.

Global schema evolution operation: Update the global schema structure.

Example 10



- Missing method

This conflict arises when there is some method in the new (exported) class that does not exist in the related global class.

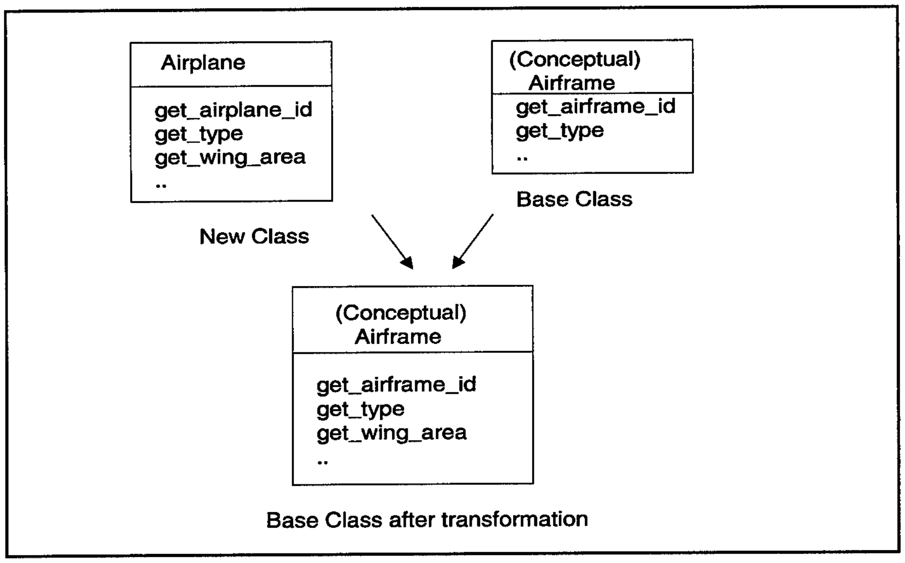
Resolution: Create the new method.

Integration Dictionary changes: The Conceptual-Method must be created in the related Conceptual-Class in the Integration Dictionary (example 11). The conflict resolution for method names must be applied and all required data must be inserted into the newly created instance.

Global schema code changes: Add the new method signature and code in the related global class schema.

Global schema evolution operation: Update the global schema structure.

Example 11



3.4.4. Schema conflict

- Missing class

This conflict arises when the concept of the new (exported) class doesn't correspond to any Conceptual-Class of the Integration Dictionary.

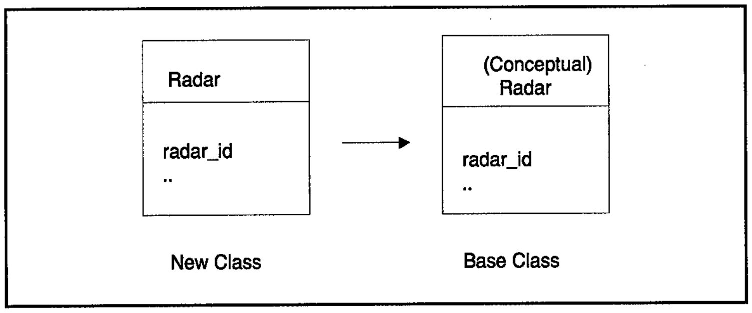
Resolution: Create a new class.

Integration Dictionary changes: The Conceptual-Class must be created in the Integration Dictionary (example 12). The conflict resolution for class names must be applied and all required data must be inserted into the newly created instance. The Real-Class must be created in the related conceptual-Class in the Integration Dictionary.

Global schema code changes: Create the new class in the global schema. After the creation of the new class schema, a pointer to the class *extent* (set that stores all the classes' instances) must be created and the pointer name must be stored in the class-root attribute of the related Real-Class object in the Integration Dictionary.

Global schema evolution operation: Update the global schema structure.

Example 12



- Missing super-class (generalization)

This conflict arises when two semantically equivalent classes (base and new) have similar and different attributes. The base class (related Conceptual-Class) does not have the necessary meaning to store the similar attributes.

Resolution: Generalize the base class defined in the global schema creating a super-class. This new super-class will contain the similar attributes of the both classes. The two classes (base and new) also have different attributes. Therefore, the base class schema will be transformed into a sub-class of this newly created super-class and will contain the different attribute(s) related only to this base class. The new (exported) class may also have different attribute(s). In this case, another sub-class class of this newly created super-class must be created to store the attribute(s) of the new class that cannot stay in the super-class (example 13).

Integration Dictionary changes: Generalize the related Conceptual-Class and Real-Class. This related Conceptual-Class only contains the different attribute(s). The similar attribute(s) in this Conceptual-Class is (are) deleted. In the example 13, the similar attributes are ir_id in Infra_Red class and radar_id in Radar class.

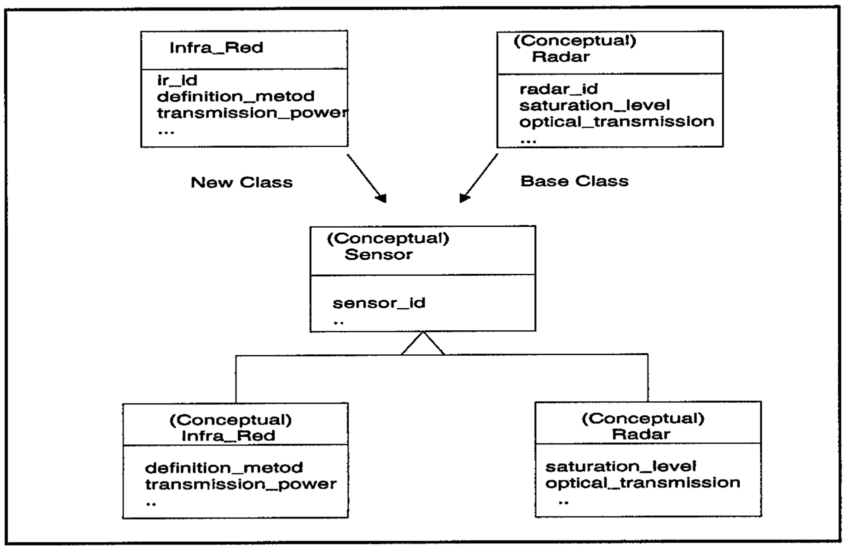
A new Conceptual-Class must be created in the Integration Dictionary to represent the super-class, which will contain the similar attribute(s). In example 13, the similar attribute (now called sensor_id) is moved to the newly created super-class. The super-conceptual class attribute of the original Conceptual-Class must be filled with the name of Conceptual-Class that represents the super-class.

If another sub-class must be created to reflect the different attributes of the new (exported) class, then another Conceptual-Class is created and will contain the different attributes of the new class. In this case, its super-conceptual class attribute must also be filled with the name of the Conceptual-Class that represents the super-class.

The conflict resolution for missing a class must be applied for the classes that must be created, and all required data must be inserted into this (these) newly created Integration Dictionary class(es).

Global Schema code changes: Generalize the related global class. Modify the related global class schema deleting the attributes and related methods that are similar. Create the global super-class schema that will contain the deleted attributes and methods. If the other sub-class (related to the different attributes of the new class) needs to be created, then create the related global sub-class schema. If the new super-class is an abstract class, the pointer to the class *extent* doesn't need to be created (the related Real-Class root-name will be null).

Global Schema evolution operation: Update the global schema structure and data migration. The data migration process moves the attribute value(s) of the original global class that is to be generalized and include it (them) into the newly created super-class.



Example 13

- Missing sub-class (specialization)

This conflict arises when two semantically equivalent classes (base and new) have similar and different attributes and the base class (related Conceptual-Class) has the necessary meaning to store the similar attributes.

Resolution: Specialize the base class defined in the global schema creating sub-classes.

Integration Dictionary changes: Specialize the Conceptual-Class and related Real-Class. The existing Conceptual-Class will contain the similar attributes. At least one new Conceptual-Class must be created to reflect the specialization of the existing Conceptual-Class (example 14).

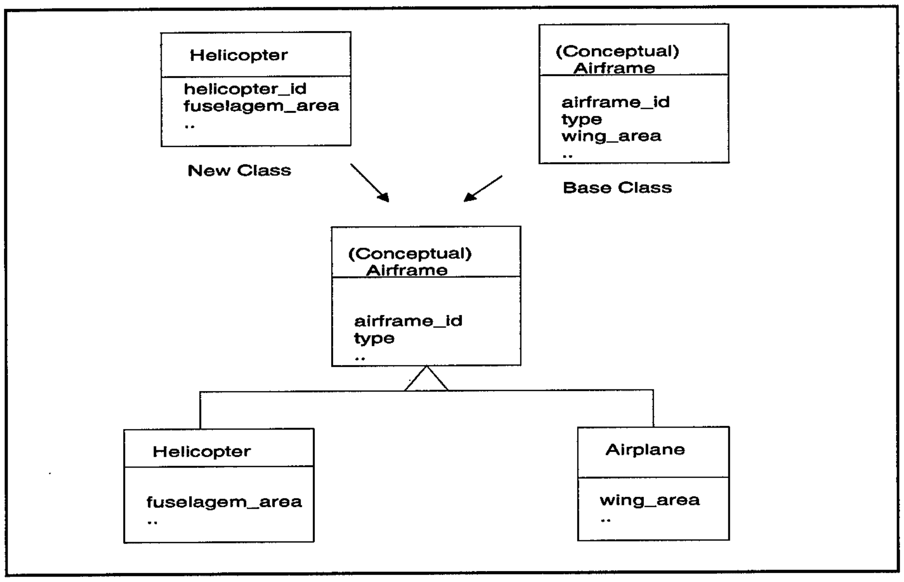
The Conceptual-Attribute(s) related to the existing Conceptual-Class that is (are) to be specialized is (are) moved to the newly created specialized Conceptual-Class.

The conflict resolution for missing class needs to be applied to the new Conceptual-Class(es), and all required conceptual data must be inserted into the newly created instance. In this case, the super-conceptual class attribute of the new Conceptual-Class(es) must be filled with the Conceptual-Class name that contains the similar attributes (super-class). The root of the new specialized class(es) must be created and if the super-class is an abstract class the related Real-Class root-name will become null.

Global schema code changes: Specialize the existing global class schema. Create the global sub-class schema(s) that will contain the different attributes and methods. Modify the specialized global class schema moving the attributes and related methods that are different the newly created sub-class schema. If the other sub-class (related to the different attributes of the new class) needs to be created, then create another global sub-class schema. If the new super-class is an abstract class, the pointer to the class *extent* doesn't need to be created (the related Real-Class root-name will be null).

Global schema evolution operation: Update the global schema structure and data migration. The data migration process moves the different attribute(s) of the original existing global class and includes it (them) into the newly created sub-class.

Example 14



- Different inheritance hierarchy

This is a more complex conflict. This conflict arises when an inheritance hierarchy from the new (exported) schema is to be integrated with a semantically related hierarchy from the base (global) schema, which has a different structure. An example of this kind of conflict is shown in Chapter 5 when the Suppressor Scenario Database schema is integrated into the global schema.

Resolution: This is a compound conflict that can be resolved by decomposing it into more primitive conflicts described in this section. Specialization, generalization, missing attribute, and missing method conflict resolution may be applied whenever necessary.

Integration Dictionary changes: The Integration Dictionary changes for specialization, generalization, missing attribute, and missing method may be applied whenever necessary.

Global schema code changes: The global schema code changes for specialization, generalization, missing attribute, and missing method may be whenever necessary.

Schema evolution operation: Update the global schema structure and data migration. The schema evolution operation defined for specialization, generalization, missing attribute and missing method may be applied depending on the schema changes.

3.5. *Schema evolution operations*

Schema evolution operations must be defined and implemented using the database language to allow update of the schema structure and data migration from the old schema to the newly modified schema. If the schema changes do require data migration, then only the schema structure must be updated to reflect the changes realized in the database schema. For each kind of schema modification an implementation solution is given, as well as the data updates. Data updates specify how existing instances of a given class in the original schema are transformed to conform to a new class definition or reclassified into the newly derived sub-class. This might involve adding or deleting attributes or aggregated objects, changing the type of a field, or deleting entire objects. Two examples of schema evolution are shown in Examples 13 and 14 where data reclassification is required. In example 13, the data stored in the attribute radar_id of the Base Class Radar must be reclassified in the newly created class Sensor. In example 14, the data stored in the attribute wing_area of the Base Class Airframe must be reclassified in the newly created class Airplane.

3.5.1. Modification to class definitions

Changes in the class structure may require schema evolution. The class structure is composed of attributes and methods. The next two sub-sections of this section describe the necessary schema evolution for class structure changes.

3.5.1.1. Attributes

- Add a new attribute.

When an attribute is added to a class, a new storage is created with this schema change and the initialization process is applied. The default initialization provided by the object-oriented database is the appropriate representation of zero.

If no default initialization is provided by the database or the required initialization of this new attribute is different from the default initialization, a schema evolution program is necessary to change the default initialization of its instances. In this case, the class with the new attribute must have an associated function that is user-supplied. This function can supply the missing initialization or override the default initialization, by generating a value for the new field.

- Deleting an attribute.

Update of the schema structure is required when the representation of a class instances is changed by removing an attribute. Since no new storage is created by this schema change, the issue of initialization does not arise.

- Change the data type of an attribute.

The schema evolution is necessary to change the representation of any of its instances by adjusting the size of the attribute's storage (if necessary) and reinitializing that storage with default values. If default initialization is provided by the object-oriented database, two types of initialization can occur. The first type is the initialization of the attribute instances with assignment-compatible values when the old and new types are compatible (e.g. the old type is integer and the new type is float). The second type is the initialization of the attribute instances with the appropriate representation of zero when the old and new types are not compatible.

3.5.1.2. Methods

Adding a new method or deleting a method has no effect on the layout of class instances, therefore schema evolution to migrate data is not necessary. But update of the schema structure is required.

3.5.2. Modification to the set of classes

- Add a new class

  This operation only requires update of the schema structure.

- Delete a class

  This operation only requires update of the schema structure.

3.5.3. Modification to inheritance hierarchy

- Adding base classes (generalization)

This schema change requires schema evolution to perform instance reclassification (data migration from the sub-class(es) to the base class). When a base class A is added to an existing class B, instances of B (and its sub-instances, if they exist) must be modified to remove the part corresponding to A.

- Removing base classes

This schema change requires schema evolution to perform instance reclassification (if the data migration from the base class to the sub-class(es) is necessary). When the base class A is deleted, its subclass B must be modified so that it no longer inherits attributes and methods from class A. Each instance of B is modified (if necessary) by adding the part corresponding to A.

- Adding sub-classes (specialization)

This schema change requires schema evolution to perform instance reclassification (data migration from the original class to a subclass). When an instance is reclassified, it acquires at least one new object derived from the original class, which contains the specialized attribute(s). The schema change also involves deleting the attribute(s) of the original class to be specialized.

3.6. Methodology for integration

In order to integrate a new schema into the global schema that cannot be directly integrated due to redundancies and anomalies, a methodology for integration must be used to guide the global schema transformation. The identification of the common and different concepts in different schemas that are mutually related to some semantic properties (scope or members) is the key to success.

This methodology proposes four steps for schema integration using the Integration Dictionary.

<u>Step 1</u>: Comparison.

During this step the exported schema is semantically compared to the global schema to identify inter-schema relations and to detect conflicts in their representation. The meaning of objects being integrated is identified based upon similar concepts stored in the integration dictionary. The similarity is manually analyzed by comparison of the exported schema classes with the conceptual classes in the Integration Dictionary.

Step 2: Conflict solving.

After the detection of conflicts, the rules for solving conflicts described in Section 3.4 and 3.5 are manually applied. During this step both exported and global schemas are brought into compatibility for later unification.

Step 3: Restructure.

This step implements the conflict resolution for schemas, which was applied in the previous step. The metadata of the global schema stored in the Integration Dictionary is updated to absorb the new integrated scenario. The Integration Dictionary also stores the relationship between the exported schema and the newly updated global schema.

Step 4: Merge.

This step unifies both the exported and the global schema structures, and their data. The data of the transformed source schema represented by the exported object-oriented schema is moved into the global schema. Figure 15 shows the integration of a new scenario into the global schema. Based on this unified global schema, all of the User's Views (exported schema) must be generated and populated before running the respective scenario generator.

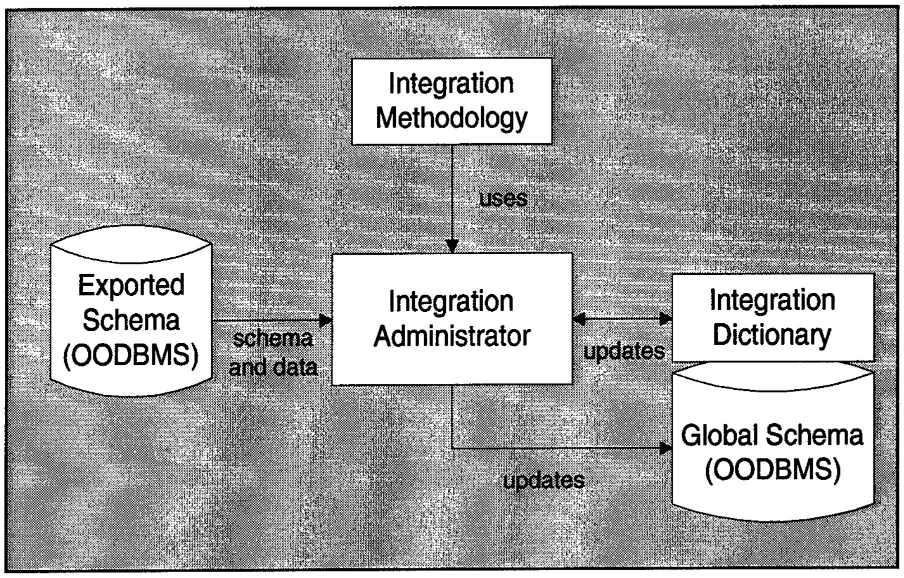


Figure 15: Integration of a New Scenario into the Global Schema

3.7. Summary

In this chapter I presented my solution for data integration of different underlying databases. The focus of this work is on scenario databases for simulation systems. My approach creates a common database repository with semantic interoperability using object-oriented technology. The Integrator Administrator must understand database systems and simulation systems in order to discover the semantic information from a source database and to perform the mappings of the exported schema and data from a specific scenario to the Integration Dictionary and common database repository, respectively. I created the Integration Dictionary in order to allow interoperability. It contains the semantics of the metadata, inter-component mappings between components of each system and the common database, and an implementation class associated with each class-component of the data dictionary. The mappings allow the generation of views of the common database, which are the specific scenario databases for simulation systems. Finally, I introduced the conflict resolutions for schema integration using the Integration Dictionary.

4. Implementation of the Prototype

4.1. Introduction

In this chapter I describe the implementation of a prototype to demonstrate the methodology for integrating databases based on the design I presented in Chapter 3. I implement this prototype using Object-Store [18]. Object-Store is an object-oriented database that provides an ideal environment for implementing this methodology. I begin the chapter by describing some important Object-Store operations that access metadata. I describe my implementation plan for the methodology in Section 4.3. Section 4.4 contains my implementation of the Integration Dictionary. In Section 4.5 the system I created to integrate metadata is described. Section 4.6 describes how I integrated exported schemas into the global schema. Section 4.7 describes the integration of data from exported schemas into the global schema. Finally in Section 4.8, I describe how the generation of Users' Views is performed.

4.2. OO-DBMS for implementation

In this thesis, Object-Store is the object-oriented database system I used to implement the global schema for scenarios of simulation systems. It has the first required extension *(extension of classes)* already embedded in its data model. This is implemented by the *extent* set. The *extent* stores all the object instances of a class.

Any change in the code of an Object-Store schema must be reflected in the system schema file. A program must be written to move the references of old schema classes to the newly modified schema.

The Object-Store database handles the following automatic changes in data, which do not involve user-defined instance initializations but instead involve default initializations.

- Adding and deleting classes.
- Adding and deleting members of classes.
- Changing the type of a class member. If the "new type" and the "old type" are assignment compatible (e.g. float type and integer type), the Object-Store assigns the value of the old class member to the new member applying standard conversions defined by the C++ language. If the "new type" and the "old type" are assignment compatible, the new member is initialized with the appropriate representation of zero.

The following operations require application-specific transformer functions.

- Renaming an attribute.
- Initializing instances of a new attribute from old or existing attribute instances.
- Moving attributes to/from a derived or base class.
- Adding or adjusting references.

These last operations need to be written in the Object-Store language in order to perform the movement of data. The data will be moved from the old class schemas to the newly transformed class schemas. More details can be found in [18].

4.3. *Implementation plan*

Figure 16 shows my general plan for implementing the methodology. The Integration Dictionary contains the metadata of the global schema. Following the methodology for integration, the Integration Administrator updates the Integration Dictionary (Restructuring step), after completing the first two steps of the methodology (Comparison and Conflict Solving). Then, the Integration Administrator updates the global schema to reflect the newly integrated exported schema. This last procedure is related to the last step of the methodology (Merge). Finally, the integration process is completed when the exported data is merged into the global schema.

The User's Views are implemented based on the exported schemas, which are generated from the global database. In order to run a specific scenario generator, the User's View must be generated (translation from the global schema to the related exported schema) and the User's View must be translated to the source scenario file (translation from the exported schema to the source schema).

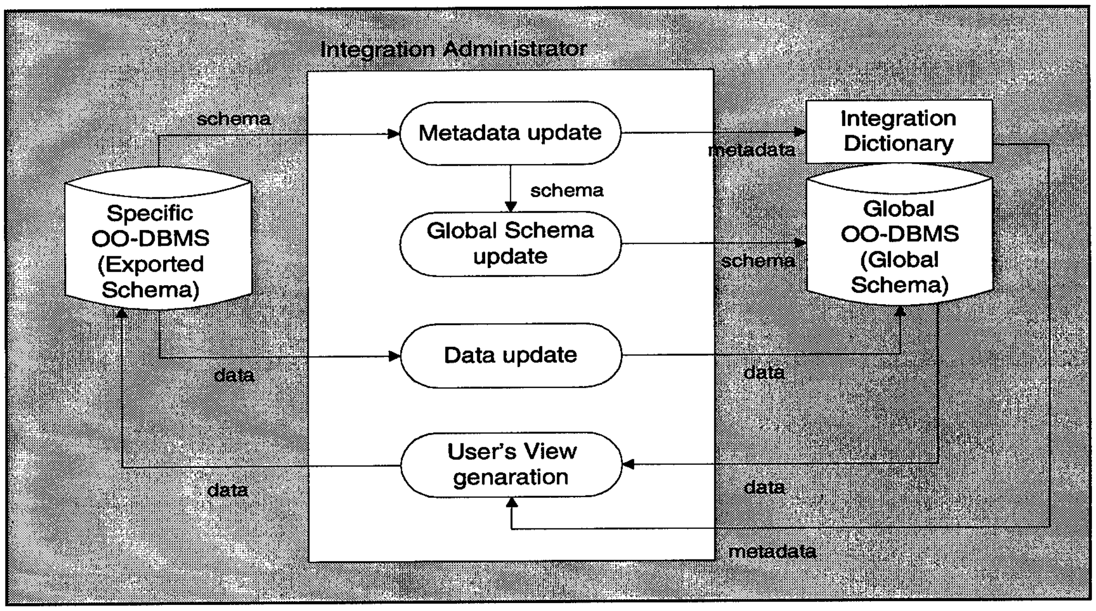


Figure 16: Implementation Plan to integrate a new scenario schema

4.3.1. Integrating a new scenario database

In Chapter 3 Section 3.2, I described the scenario's integration. In this section I introduce a step-by-step guide to be applied by the IA when performing the integration process.

Assuming that the schema translation was already performed from the source scenario file to the OO-Model (exported schema), the first schema (base schema) is integrated to the empty global database. In this case the first two steps of the methodology are not applied (comparison and conflict solving). The concepts of the first exported schema are inserted into the Integration Dictionary and the global schema will have the same schema structure as this exported schema.

The following steps of the integration methodology must be applied to integrate a new scenario database into the non-empty global database.

- The IA **compares** the exported schema to the global schema using the concepts stored into the Integration Dictionary, and detects the conflicts and similarities between both schemas.
- The IA **solves the conflicts** detected in the previous step, applying the conflict resolution defined in Chapter 3, Section 3.4 - *Resolution.*
- During this step the IA implements the resolution by applying the Integration Dictionary changes defined in Chapter 3, Section 3.4 – *Integration Dictionary changes*, **restructuring** the metadata information through the Integration Dictionary interface system (described in Section 4.5) and creating the relations (mappings) between both schemas.

- In this last step the IA **merges** both schemas by transforming the global schema structure (as defined in Chapter 3, Section 3.4 – *Global schema changes* and *Global schema evolution*) to reflect the newly integrated schema; and by merging their data. The IA must write the necessary modification in the global schema classes. In order to merge data the IA must solve the data conflicts (if detected) and write an Object-Store program using methods defined in the exported schema to read the data and constructor methods defined in the global schema to write the data.

4.3.2. Generating a specific User's View

The global database contains integrated data of all simulation system components. In order to run a simulation system, a scenario must be specified and created. The first step to create this scenario is to generate the specific User's View. This requires a translation program for the specific User's View. The second step is to translate the User's View to the source scenario file. This second step is not in the scope of this thesis but it was investigated by Weber [32].

To create the User's View translation program, the IA must follow the steps below:

- Run the View Constructor program (discussed in Section 4.8), which gives the exported class names to be generated as well as the methods of the related global classes (classes of the global database) that access the global attributes needed as parameters for constructing the classes of the specific User's View.
- Write an Object-Store program (User's View translation program) using the constructor methods defined for the classes of the User's View (the class names were

given by the View Constructor program and are defined in the exported schema) to insert new instances into the User's View database. The parameters of specific User's View constructor methods are attribute values stored into the related classes. These attribute values are accessed by the read methods of the global schema also given by the View Constructor program.

4.4. Implementation of the Integration Dictionary

The Integration Dictionary classes were created based on the integration dictionary model described in Chapter 3. They store the metadata information of the global schema. Below, the header codes of all the Integration Dictionary classes' schema are described.

<u>Conceptual-Class:</u>

The attribute Conceptual_RealClass is implemented as a pointer to the respective aggregate object. The attributes Conceptual_ClassAliases, Conceptual_SuperClasses, Conceptual_Attributes and Conceptual_Methods are implemented, respectively, as a set of pointers to Classalias, Conceptual_Classes, Conceptual_Attributes and Conceptual_Methods aggregate set of objects.

Conceptual-Class header

```
extern os_Set<conceptual_class*> *conceptual_class_extent;

class conceptual_class
{
public:
     const char* Class_GetName() {return Conceptual_ClassName;}
     const char* Class_GetSuperClassName();

     real_class* Class_GetRealClass() {return Conceptual_RealClass;}
     void Class_SetRealClass(real_class* rclass);

     os_Set<conceptual_class*> Class_GetSuperClasses() { return Conceptual_SuperClasses;
}
     void Class_RemoveSuperClass(conceptual_class* cclass)  {
Conceptual_SuperClasses.remove(cclass); }
     void Class_InsertSuperClass(conceptual_class* cclass);
     os_Set<conceptual_attribute*> Class_GetAttributes() { return Conceptual_Attributes;
}
     void Class_RemoveAttribute(conceptual_attribute* attribute_name)  {
Conceptual_Attributes.remove(attribute_name); }
     void Class_InsertAttribute(conceptual_attribute* attribute_name);

     os_Set<conceptual_method*> Class_GetMethods() { return Conceptual_Methods; }
     void Class_RemoveMethod(conceptual_method* method_name) {
Conceptual_Methods.remove(method_name); }
     void Class_InsertMethod(conceptual_method* method_name);

     os_Set<classalias*> Class_GetAlias() { return Conceptual_ClassAliases; }
     void Class_RemoveAlias(char* alias_name);
     void Class_InsertAlias(classalias* alias_name);

     void Class_Setclass(const char* class_descrp, conceptual_class* class_superclass);
     conceptual_class(const char* class_name, const char* class_descrp, conceptual_class*
superclass);
     void Class_Show(ostream& os);
     static os_typespec* get_os_typespec();
     ~conceptual_class();

private:
     char Conceptual_ClassName[51];
     char Conceptual_ClassDescription[201];
     os_Set<conceptual_class*> Conceptual_SuperClasses;
     os_Set<classalias*> Conceptual_ClassAliases;
     os_Set<conceptual_attribute*> Conceptual_Attributes;
     os_Set<conceptual_method*> Conceptual_Methods;
     real_class* Conceptual_RealClass;
};
```

Conceptual-Attribute:

The attribute Conceptual_AttributeAliases is implemented as a set of pointers to Attralias aggregate objects.

Conceptual-Class header

```
class conceptual_attribute
{
public:
     const char* Attribute_GetUnit() {return Conceptual_UnitofRepresentation;}
     const char* Attribute_GetName() {return Conceptual_AttributeName;}
     char Attribute_GetAttributeType() {return Conceptual_AttributeType;}
     const char* Attribute_GetTypeName() {return Conceptual_TypeName;}

     os_Set<attralias*> Attribute_GetAlias() { return Conceptual_AttributeAliases; }
     void Attribute_RemoveAlias(char* alias_name);
     void Attribute_InsertAlias(attralias* alias_name);

     void Attribute_Setattribute( const char* attr_description, const char* attr_unit,
char attr_type, const char* attr_typename, int attr_sz);
     conceptual_attribute(const char* attr_name, const char* attr_description, const
char* attr_unit, char attr_type, const char* attr_typename, int attr_sz);
     ~conceptual_attribute();
     void Attribute_Show(ostream& os);
     static os_typespec* get_os_typespec();

private:
     char Conceptual_AttributeName[51];
     char Conceptual_AttributeDescription[201];
     char Conceptual_UnitofRepresentation[21];
     char Conceptual_AttributeType;
     char Conceptual_TypeName[31];
     int Conceptual_AttributeSize;
     os_Set<attralias*> Conceptual_AttributeAliases;

};
```

Conceptual-Method:

The attribute Conceptual_MethodAliases is implemented as a set of pointers to Methalias aggregate objects.

Conceptual-Method header

```
class conceptual_method
{
public:
     const char* Method_GetName() {return Conceptual_MethodName;}
     int Method_GetType() {return Conceptual_MethodType;}

     os_Set<methalias*> Method_GetAlias() { return Conceptual_MethodAliases; }
     void Method_RemoveAlias(char* alias_name);
     void Method_InsertAlias(methalias* alias_name);

     void Method_Show(ostream& os);
     void Method_Setmethod(int mtype, char* meth_description);
     conceptual_method(char* meth_name, int mtype, char* meth_description);
     ~conceptual_method();
     static os_typespec* get_os_typespec();
private:
     char Conceptual_MethodName[51];
     int Conceptual_MethodType;
     char Conceptual_MethodDescription[201];
     os_Set<methalias*> Conceptual_MethodAliases;
};
```

Real Class:

Real-Class header

```
class real_class
{
public:
     const char* RealClass_GetClassRoot() {return Real_ClassRoot;}
     const char* RealClass_GetName() { return Real_ClassName; }
     void RealClass_Setrealclass(char* root_name, char* struct_file, char*
complementary_file);
     real_class(char* class_name, char* root_name, char* struct_file, char*
complementary_file);
     ~real_class();
     void RealClass_Show(ostream& os);
     static os_typespec* get_os_typespec();

private:
     char Real_ClassName[51];
     char Real_ClassRoot[51];
     char Real_ClassStructureFile[51];
     char Real_ClassComplementaryFile[51];
};
```

Alias:

This class was split into three classes in order to allow the implementation of aggregate objects with two-way pointers. Two-way pointers facilitate the user's view implementation since the programmer has the flexibility of accessing the related Conceptual-Class, Conceptual-Attribute or Conceptual-Method given a respective alias name. Each alias class has an attribute that points back to the aggregated class. This means that an instance of any of the three alias classes will point back to the instance of the related Conceptual-Class, Conceptual-Attribute or Conceptual-Method instance from which it is aggregated.

- Classalias

The attribute Alias_RelatedClass points back to an instance of the Conceptual-Class.

Classalias header

```
class classalias
{
public:
     char* Classalias_GetAlias() {return Alias_Class;}
     classalias(const char* alias_name, conceptual_class* classnm);
     ~classalias();
     static os_typespec* get_os_typespec();

private:
     char Alias_Class[61];
     conceptual_class* Alias_RelatedClass;
};
```

- <u>Attralias</u>

The attribute Alias_RelatedAttribute points back to an instance of the Conceptual-Attribute class.

Attralias header

```
class attralias
{
public:
     char* Attralias_GetAlias() {return Alias_Attribute;}
     conceptual_attribute* Attralias_GetRealatedAttribute() {return
Alias_RelatedAttribute;}
     attralias(const char* alias_name, conceptual_attribute* attrnm);
     ~attralias();
     static os_typespec* get_os_typespec();

private:
     char Alias_Attribute[61];
     conceptual_attribute* Alias_RelatedAttribute;
};
```

- <u>Methalias</u>

The attribute Alias_RelatedMethod points back to an instance of the Conceptual-Method class.

Methalias header

```
class methalias
{
public:
     char* Methalias_GetAlias() {return Alias_Method;}
     conceptual_method* Methalias_GetRealatedMethod() {return  Alias_RelatedMethod;}
     methalias(const char* alias_name, conceptual_method* methnm);
     ~methalias();
     static os_typespec* get_os_typespec();

private:
     char Alias_Method[61];
     conceptual_method* Alias_RelatedMethod;
};
```

4.5. Metadata integration

4.5.1. System structure

I created a system to implement metadata integration. This system updates the metadata stored in the Integration Dictionary through a user's interface. It uses methods defined for each class of the Integration Dictionary schema. The insertion process for the Integration Dictionary components (Conceptual-Classes, Conceptual-Attributes, Conceptual-Methods and Real-Classes) can read input data stored in text files (when more than one instance needs to be inserted) or typed directly from the keyboard.

4.5.2. Conflict X System Functions

Below I list the system functions to be applied for each possible conflict (discussed in Chapter 3) that can arise from the integration of two schemas.

Table 1: Conflict X System Functions

Conflict	**Functions to be applied**
Class Name	Change Alias of a Class/Insert
Attribute Name	Change Alias of an Attribute/Insert
Method Name	Change Alias of a Method/Insert
Expressions	A Dictionary Method must be created* Insert a Method
Units	A Dictionary Method must be created* Insert a Method
Precision	A Dictionary Method must be created* Insert a Method
Attribute Composition	Insert a Method
Attribute Data Type	A Dictionary Method must be created* Insert a Method
Attribute Concatenation	Insert a Method
Missing Attribute	Insert an attribute
Missing Method	Insert a Method
Missing Class	Insert a Class Insert a Real Class

Missing Super-Class (Generalization)	- Change Attribute of a Class/ Remove (remove similar attributes from the existing sub-class) - Insert a Class (super-class class that will hold the similar attributes) - Change Attributes of a Class/Insert (insert the previously removed attributes into the newly created super-class) - Change Super-Classes/Insert (insert the newly created super-class name into the super-classes set of the sub-class) - Change Real Class/Insert (set the real-class attribute of the newly created super-class to the same real-class as stored in the real-class attribute of the sub-class)
Missing Sub-Class (Specialization)	- Change Attribute of a Class/ Remove (remove different attributes from the existing super-class) - Insert a Class (sub-class that will hold the different attributes, the attribute super-class will contain the name of the existing super-class) - Change Attributes of a Class/Insert (insert the previously removed attributes into the newly created sub-class) - Change Real Class/Insert (set the real-class attribute of the newly created sub-class to the same real-class as stored in the real-class attribute of the super-class)

*Dictionary Methods are written in separate files: dictionary.hh for signatures and dictionary.cc for implementation code.

4.6. Schema integration

The IA must create specific programs to allow schema evolution (if necessary) when schema integration is performed. These programs can use the pre-defined functions described in Section 4.2 or new functions defined by the Integration Administrator. There are two cases of schema evolution. The first involves schema modification which just requires update of the schema structure. The second involves schema modification followed by data modification which requires update of the schema structure and data migration. The second case is applied when:

- changes in a hierarchy structure are necessary;

- initialization of instances of a new attribute is made based on old or existing attribute instances; and
- automatic transformation of values cannot be applied when the type of a class member is changed.

4.7. Data integration

Specific programs must be created to implement data integration. These programs are the mappings between the exported schema and the global schema. They must be created for each simulation system component of the global schema in order to integrate their instances into the global schema. Data conflict must be solved by the IA and, if necessary, the data administrator of the system component.

4.8. User's View generation

Specific programs must be written to create the User's Views. These programs are mappings between the global schema and the exported schemas. They use metadata stored in the Integration Dictionary as well as data stored in the global schema to generate the related view. These views must be generated for each simulation system component of the global schema before running the respective scenario generator. The View Constructor program was created to help the Integrator Administrator (IA) with the last task for completion of the schema integration – the creation of the User's View for each component system. In order to be able to use this tool, each exported schema must have a naming rule that includes, for each component of the schema, the corresponding system-code located in the three first characters of its name

The IA must input the system-code of the component system for the specific view generation (for example EAD for EADSIM). The system shows all the Conceptual-Classes related to the User's View. For each Conceptual-Class the program shows the information listed below.

- The related exported class name.
- The access methods of the global schema used to construct instances of the exported class. These methods are used as parameters of the exported class constructor method.
- All the attribute values stored in the real class of the Integrator Dictionary: real class name and real class root. These values are used to reference the global class in the User's View translation program.
- The construction strategy for aggregated attributes. The program outputs information about the attributes that are multi-valued objects and single objects. It also outputs the classes of these objects which must be generated before the class(es) that contains the aggregated objects.
- In the case where the generation of instances of an exported class come from two or more global classes (join operation), the program shows the same exported class name for these global classes.

4.9. Summary

In this chapter I described my implementation plan used to perform database integration. I discussed each task of this plan. The first task is to update the metadata. I created a system tool to do this. For the second task (integrate schema code) no specific tool was designed since the code must be directly modified using the specific OO-DBMS definition language and any specific case of schema evolution must be written in the specific OO-DBMS manipulation language. Also, no specific tool was designed for the third task (integrate data) since it needs specific programs in order to perform data integration. Finally, for the fourth task (generating User's Views) I designed a tool to show the IA how to construct a specific view. In the next chapter, I integrate two simulation system scenarios into the global schema to validate this prototype and the integration methodology.

5. Validation of the Methodology

5.1. Introduction

Two simulation systems are used to validate my methodology for database integration: EADSIM and Suppressor, both described in Chapter 2. All steps necessary for integration, described in Chapter 4, are applied in both systems' integration. Also, all specific programs for data and schema integration and User's View generation are written for each one of the systems. The scope for integration is limited to a subset of the respective systems to allow a demonstration of the new methodology within the available time limits. This definition is focused on basic elements of the simulation system's scenario. The data used to demonstrate the validation is fictitious. In Section 5.2 I describe the EADSIM scenario integration and in Section 5.3 I describe the Suppressor scenario integration. In Section 5.4 I examine if the data examples were successfully inserted in the global database.

5.2. Integrating EADSIM

The following sub-sections of this section describe all the steps used to integrate the EADSIM Scenario Database (OO-DBMS) into the global database.

5.2.1. Scope for integration

Part of the EADSIM Scenario Database schema was selected to demonstrate the integration. In Figure 17 the system Object Model is shown. A system aggregates basic elements, as well as other systems.

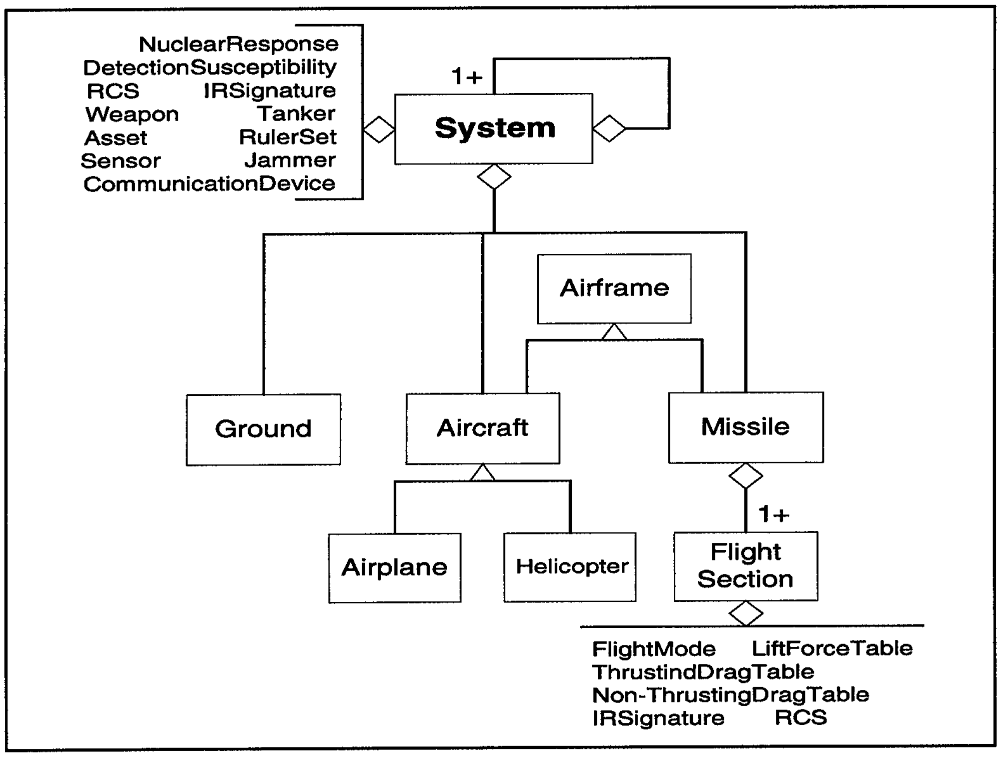


Figure 17: EADSIM Object Model

In the system scope, three basic elements were chosen to be implemented in the global schema example: Airplane, Helicopter and Missile (the Missile aggregates are not in the scope). The scope for implementation is shown in Figure 18.

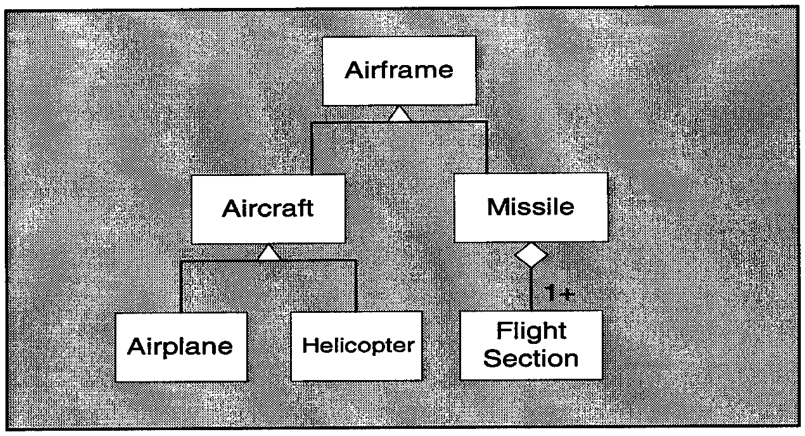


Figure 18: EADSIM Object Model Scope for Implementation

5.2.2. Definition of the exported schema

For each class shown in Figure 17, I created the implementation code using the Object-Store schema definition language. According to Section 4.7, the naming rule must be applied to allow view generation. All schema components have EAD in the three first characters of their names.

Also, only the lower level classes of the hierarchy (Airplane, Helicopter and Missile) were implemented as concrete classes. The body codes of the EADSIM exported classes' schema are shown in Appendix B, Section 1. Below I present the header code of the classes' schema.

Class EAD_Airframe

EAD_Airframe header

```
class EAD_Airframe
{
public:
     // Get Methods
     char* EAD_Get_Airframe_ID() {return EAD_Airframe_ID;}

     // Set Methods
     void EAD_Set_Airframe_ID(char* ID) {strcpy(EAD_Airframe_ID, ID);}

     // Constructor and Destructor Methods
     EAD_Airframe(char* Airframe_ID);
     ~EAD_Airframe();

     static os_typespec* get_os_typespec();

protected:
     char EAD_Airframe_ID[31];
};
```

Class EAD_Aircraft

EAD_Aircraft header

```
class EAD_Aircraft : EAD_Airframe
{
public:
     // Get Method calls parent
     char* EAD_Get_Aircraft_ID() {return EAD_Airframe::EAD_Get_Airframe_ID();}
     // Get Methods
     int EAD_Get_Aircraft_NonAerodynamic() {return EAD_Aircraft_NonAerodynamic;}
     int EAD_Get_Aircraft_MAXSpeed() {return EAD_Aircraft_MAXSpeed;}
     int EAD_Get_Aircraft_MINSpeed() {return EAD_Aircraft_MINSpeed;}
     float EAD_Get_Aircraft_MAXG() {return EAD_Aircraft_MAXG;}
     int EAD_Get_Aircraft_EmptyWeight() {return EAD_Aircraft_EmptyWeight;}
     int EAD_Get_Aircraft_FuelWeight() {return EAD_Aircraft_FuelWeight;}
     int EAD_Get_Aircraft_RTBFuelBingoLimit() {return EAD_Aircraft_RTBFuelBingoLimit;}

     int EAD_Get_Aircraft_AirRefuelBingoLimit() {return
EAD_Aircraft_AirRefuelBingoLimit;}
     int EAD_Get_Aircraft_MaxFuelReceivingRate() {return
EAD_Aircraft_MaxFuelReceivingRate;}
     float EAD_Get_Aircraft_LookAheadInterval() {return EAD_Aircraft_LookAheadInterval;}
     float EAD_Get_Aircraft_MultiplicationFactor() {return
EAD_Aircraft_MultiplicationFactor;}
     int EAD_Get_Aircraft_TerrainSamples() {return EAD_Aircraft_TerrainSamples;}
     int EAD_Get_Aircraft_FeedbackControlGain() {return
EAD_Aircraft_FeedbackControlGain;}
     int EAD_Get_Aircraft_MAXClimbAngle() {return EAD_Aircraft_MAXClimbAngle;}

     // Set Methods from the parent
     void EAD_Set_Aircraft_ID(char* ID) {EAD_Airframe::EAD_Set_Airframe_ID(ID);}

     // Set Methods
     void EAD_Set_Aircraft_NonAerodynamic(int NonAerodynamic)
{EAD_Aircraft_NonAerodynamic = NonAerodynamic;}
     void EAD_Set_Aircraft_MAXSpeed(int MAXSpeed) {EAD_Aircraft_MAXSpeed = MAXSpeed;}
     void EAD_Set_Aircraft_MINSpeed(int MINSpeed) {EAD_Aircraft_MINSpeed = MINSpeed;}
     void EAD_Set_Aircraft_MAXG(float MAXG) {EAD_Aircraft_MAXG = MAXG;}
     void EAD_Set_Aircraft_EmptyWeight(int EmptyWeight) {EAD_Aircraft_EmptyWeight =
EmptyWeight;}
     void EAD_Set_Aircraft_FuelWeight(int FuelWeight) {EAD_Aircraft_FuelWeight =
FuelWeight;}
     void EAD_Set_Aircraft_RTBFuelBingoLimit(int RTBFuelBingoLimit)
{EAD_Aircraft_RTBFuelBingoLimit = RTBFuelBingoLimit;}
     void EAD_Set_Aircraft_AirRefuelBingoLimit(int AirRefuelBingoLimit)
{EAD_Aircraft_AirRefuelBingoLimit = AirRefuelBingoLimit;}
     void EAD_Set_Aircraft_MaxFuelReceivingRate(int MaxFuelReceivingRate)
{EAD_Aircraft_MaxFuelReceivingRate = MaxFuelReceivingRate;}
     void EAD_Set_Aircraft_LookAheadInterval(float LookAheadInterval)
{EAD_Aircraft_LookAheadInterval = LookAheadInterval;}
     void EAD_Set_Aircraft_MultiplicationFactor(float MultiplicationFactor)
{EAD_Aircraft_MultiplicationFactor = MultiplicationFactor;}
     void EAD_Set_Aircraft_TerrainSamples(int TerrainSamples)
{EAD_Aircraft_TerrainSamples = TerrainSamples;}
     void EAD_Set_Aircraft_FeedbackControlGain(int FeedbackControlGain)
{EAD_Aircraft_FeedbackControlGain = FeedbackControlGain;}
     void EAD_Set_Aircraft_MAXClimbAngle(int MAXClimbAngle) {EAD_Aircraft_MAXClimbAngle =
MAXClimbAngle;}

     // Constructor and Destructor Methods
     EAD_Aircraft(char* Airframe_ID, int NonAerodynamic, int MAXSpeed, int MINSpeed,
float MAXG, int EmptyWeight, int FuelWeight, int RTBFuelBingoLimit, int
AirRefuelBingoLimit, int MaxFuelReceivingRate, float LookAheadInterval, float
MultiplicationFactor, int TerrainSamples, int FeedbackControlGain, int MAXClimbAngle);
     ~EAD_Aircraft();

     static os_typespec* get_os_typespec();

protected:
     int EAD_Aircraft_NonAerodynamic;
     int EAD_Aircraft_MAXSpeed;
     int EAD_Aircraft_MINSpeed;
     float EAD_Aircraft_MAXG;
     int EAD_Aircraft_EmptyWeight;
     int EAD_Aircraft_FuelWeight;
     int EAD_Aircraft_RTBFuelBingoLimit;
     int EAD_Aircraft_AirRefuelBingoLimit;
     int EAD_Aircraft_MaxFuelReceivingRate;
     float EAD_Aircraft_LookAheadInterval;
     float EAD_Aircraft_MultiplicationFactor;
     int EAD_Aircraft_TerrainSamples;
     int EAD_Aircraft_FeedbackControlGain;
     int EAD_Aircraft_MAXClimbAngle;
};
```

Class EAD_Airplane

EAD_Airplane header

```
class EAD_Airplane : EAD_Aircraft
{
public:
     // Get Methods call parent
     char* EAD_Get_Airplane_ID() {return EAD_Aircraft::EAD_Get_Aircraft_ID();}
     int EAD_Get_Airplane_NonAerodynamic() {return
EAD_Aircraft::EAD_Get_Aircraft_NonAerodynamic();}
     int EAD_Get_Airplane_MAXSpeed() {return EAD_Aircraft::EAD_Get_Aircraft_MAXSpeed();}
     int EAD_Get_Airplane_MINSpeed() {return EAD_Aircraft::EAD_Get_Aircraft_MINSpeed();}
     float EAD_Get_Airplane_MAXG() {return EAD_Aircraft::EAD_Get_Aircraft_MAXG();}
     int EAD_Get_Airplane_EmptyWeight() {return
EAD_Aircraft::EAD_Get_Aircraft_EmptyWeight();}
     int EAD_Get_Airplane_FuelWeight() {return
EAD_Aircraft::EAD_Get_Aircraft_FuelWeight();}
     int EAD_Get_Airplane_RTBFuelBingoLimit() {return
EAD_Aircraft::EAD_Get_Aircraft_RTBFuelBingoLimit();}
     int EAD_Get_Airplane_AirRefuelBingoLimit() {return
EAD_Aircraft::EAD_Get_Aircraft_AirRefuelBingoLimit();}
     int EAD_Get_Airplane_MaxFuelReceivingRate() {return
EAD_Aircraft::EAD_Get_Aircraft_MaxFuelReceivingRate();}
     float EAD_Get_Airplane_LookAheadInterval() {return
EAD_Aircraft::EAD_Get_Aircraft_LookAheadInterval();}
     float EAD_Get_Airplane_MultiplicationFactor() {return
EAD_Aircraft::EAD_Get_Aircraft_MultiplicationFactor();}
     int EAD_Get_Airplane_TerrainSamples() {return
EAD_Aircraft::EAD_Get_Aircraft_TerrainSamples();}
     int EAD_Get_Airplane_FeedbackControlGain() {return
EAD_Aircraft::EAD_Get_Aircraft_FeedbackControlGain();}
     int EAD_Get_Airplane_MAXClimbAngle() {return
EAD_Aircraft::EAD_Get_Aircraft_MAXClimbAngle();}

     // Get Methods
     int EAD_Get_Airplane_ABSpeed() {return EAD_Airplane_ABSpeed;}
     int EAD_Get_Airplane_MAXThrust() {return EAD_Airplane_MAXThrust;}
     float EAD_Get_Airplane_WingArea() {return EAD_Airplane_WingArea;}
     float EAD_Get_Airplane_WingSpan() {return EAD_Airplane_WingSpan;}
     int EAD_Get_Airplane_CruiseAltMSL() {return EAD_Airplane_CruiseAltMSL;}
     int EAD_Get_Airplane_CruiseSpeed() {return EAD_Airplane_CruiseSpeed;}
     int EAD_Get_Airplane_FuelFlow() {return EAD_Airplane_FuelFlow;}
     float EAD_Get_Airplane_TSFC() {return EAD_Airplane_TSFC;}
     int EAD_Get_Airplane_MAXSpeedCruiseAlt() {return EAD_Airplane_MAXSpeedCruiseAlt;}
     float EAD_Get_Airplane_CD0() {return EAD_Airplane_CD0;}

     // Set Methods from the parent
     void EAD_Set_Airplane_ID(char* ID) {EAD_Aircraft::EAD_Set_Aircraft_ID(ID);}
     void EAD_Set_Airplane_NonAerodynamic(int NonAerodynamic)
{EAD_Aircraft::EAD_Set_Aircraft_NonAerodynamic(NonAerodynamic);}
     void EAD_Set_Airplane_MAXSpeed(int MAXSpeed)
{EAD_Aircraft::EAD_Set_Aircraft_MAXSpeed(MAXSpeed);}
     void EAD_Set_Airplane_MINSpeed(int MINSpeed)
{EAD_Aircraft::EAD_Set_Aircraft_MINSpeed(MINSpeed);}
     void EAD_Set_Airplane_MAXG(float MAXG) {EAD_Aircraft::EAD_Set_Aircraft_MAXG(MAXG);}
     void EAD_Set_Airplane_EmptyWeight(int EmptyWeight)
{EAD_Aircraft::EAD_Set_Aircraft_EmptyWeight(EmptyWeight);}
```

```
     void EAD_Set_Airplane_FuelWeight(int FuelWeight)
{EAD_Aircraft::EAD_Set_Aircraft_FuelWeight(FuelWeight);}
     void EAD_Set_Airplane_RTBFuelBingoLimit(int RTBFuelBingoLimit)
{EAD_Aircraft::EAD_Set_Aircraft_RTBFuelBingoLimit(RTBFuelBingoLimit);}
     void EAD_Set_Airplane_AirRefuelBingoLimit(int AirRefuelBingoLimit)
{EAD_Aircraft::EAD_Set_Aircraft_AirRefuelBingoLimit(AirRefuelBingoLimit);}
     void EAD_Set_Airplane_MaxFuelReceivingRate(int MaxFuelReceivingRate)
{EAD_Aircraft::EAD_Set_Aircraft_MaxFuelReceivingRate(MaxFuelReceivingRate);}
     void EAD_Set_Airplane_LookAheadInterval(float LookAheadInterval)
{EAD_Aircraft::EAD_Set_Aircraft_LookAheadInterval(LookAheadInterval);}
     void EAD_Set_Airplane_MultiplicationFactor(float MultiplicationFactor)
{EAD_Aircraft::EAD_Set_Aircraft_MultiplicationFactor(MultiplicationFactor);}
     void EAD_Set_Airplane_TerrainSamples(int TerrainSamples)
{EAD_Aircraft::EAD_Set_Aircraft_TerrainSamples(TerrainSamples);}
     void EAD_Set_Airplane_FeedbackControlGain(int FeedbackControlGain)
{EAD_Aircraft::EAD_Set_Aircraft_FeedbackControlGain(FeedbackControlGain);}
     void EAD_Set_Airplane_MAXClimbAngle(int MAXClimbAngle)
{EAD_Aircraft::EAD_Set_Aircraft_MAXClimbAngle(MAXClimbAngle);}

     // Set Methods
     void EAD_Set_Airplane_ABSpeed(int ABSpeed) {EAD_Airplane_ABSpeed = ABSpeed;}
     void EAD_Set_Airplane_MAXThrust(int MAXThrust) {EAD_Airplane_MAXThrust = MAXThrust;}
     void EAD_Set_Airplane_WingArea(float WingArea) {EAD_Airplane_WingArea = WingArea;}
     void EAD_Set_Airplane_WingSpan(float WingSpan) {EAD_Airplane_WingSpan = WingSpan;}
     void EAD_Set_Airplane_CruiseAltMSL(int CruiseAltMSL) {EAD_Airplane_CruiseAltMSL =
CruiseAltMSL;}
     void EAD_Set_Airplane_CruiseSpeed(int CruiseSpeed) {EAD_Airplane_CruiseSpeed =
CruiseSpeed;}
     void EAD_Set_Airplane_FuelFlow(int FuelFlow) {EAD_Airplane_FuelFlow = FuelFlow;}
     void EAD_Set_Airplane_TSFC(float TSFC) {EAD_Airplane_TSFC = TSFC;}
     void EAD_Set_Airplane_MAXSpeedCruiseAlt(int MAXSpeedCruiseAlt)
{EAD_Airplane_MAXSpeedCruiseAlt = MAXSpeedCruiseAlt;}
     void EAD_Set_Airplane_CD0(float CD0) {EAD_Airplane_CD0 = CD0;}

     // Constructor and Destructor Methods
     EAD_Airplane(char* Airframe_ID, int NonAerodynamic, int MAXSpeed, int MINSpeed,
float MAXG, int EmptyWeight, int FuelWeight, int RTBFuelBingoLimit, int
AirRefuelBingoLimit, int MaxFuelReceivingRate, float LookAheadInterval, float
MultiplicationFactor, int TerrainSamples, int FeedbackControlGain, int MAXClimbAngle, int
ABSpeed, int MAXThrust, float WingArea, float WingSpan, int CruiseAltMSL, int
CruiseSpeed, int FuelFlow, float TSFC, int MAXSpeedCruiseAlt, float CD0);
     ~EAD_Airplane();

     static os_typespec* get_os_typespec();

private:
     int EAD_Airplane_ABSpeed;
     int EAD_Airplane_MAXThrust;
     float EAD_Airplane_WingArea;
     float EAD_Airplane_WingSpan;
     int EAD_Airplane_CruiseAltMSL;
     int EAD_Airplane_CruiseSpeed;
     int EAD_Airplane_FuelFlow;
     float EAD_Airplane_TSFC;
     int EAD_Airplane_MAXSpeedCruiseAlt;
     float EAD_Airplane_CD0;
};
```

Class EAD_Helicopter

EAD_Helicopter header

```
class EAD_Helicopter : EAD_Aircraft
{
public:
     // Get Methods call parent
     char* EAD_Get_Helicopter_ID() {return EAD_Aircraft::EAD_Get_Aircraft_ID();}
     int EAD_Get_Helicopter_NonAerodynamic() {return
EAD_Aircraft::EAD_Get_Aircraft_NonAerodynamic();}
     int EAD_Get_Helicopter_MAXSpeed() {return
EAD_Aircraft::EAD_Get_Aircraft_MAXSpeed();}
     int EAD_Get_Helicopter_MINSpeed() {return
EAD_Aircraft::EAD_Get_Aircraft_MINSpeed();}
     float EAD_Get_Helicopter_MAXG() {return EAD_Aircraft::EAD_Get_Aircraft_MAXG();}
     int EAD_Get_Helicopter_EmptyWeight() {return
EAD_Aircraft::EAD_Get_Aircraft_EmptyWeight();}
     int EAD_Get_Helicopter_FuelWeight() {return
EAD_Aircraft::EAD_Get_Aircraft_FuelWeight();}
     int EAD_Get_Helicopter_RTBFuelBingoLimit() {return
EAD_Aircraft::EAD_Get_Aircraft_RTBFuelBingoLimit();}
     int EAD_Get_Helicopter_AirRefuelBingoLimit() {return
EAD_Aircraft::EAD_Get_Aircraft_AirRefuelBingoLimit();}
     int EAD_Get_Helicopter_MaxFuelReceivingRate() {return
EAD_Aircraft::EAD_Get_Aircraft_MaxFuelReceivingRate();}
     float EAD_Get_Helicopter_LookAheadInterval() {return
EAD_Aircraft::EAD_Get_Aircraft_LookAheadInterval();}
     float EAD_Get_Helicopter_MultiplicationFactor() {return
EAD_Aircraft::EAD_Get_Aircraft_MultiplicationFactor();}
     int EAD_Get_Helicopter_TerrainSamples() {return
EAD_Aircraft::EAD_Get_Aircraft_TerrainSamples();}
     int EAD_Get_Helicopter_FeedbackControlGain() {return
EAD_Aircraft::EAD_Get_Aircraft_FeedbackControlGain();}
     int EAD_Get_Helicopter_MAXClimbAngle() {return
EAD_Aircraft::EAD_Get_Aircraft_MAXClimbAngle();}

     // Get Methods
     float EAD_Get_Helicopter_Power() {return EAD_Helicopter_Power;}
     float EAD_Get_Helicopter_PSFC() {return EAD_Helicopter_PSFC;}
     float EAD_Get_Helicopter_DecelarationG() {return EAD_Helicopter_DecelarationG;}
     int EAD_Get_Helicopter_Blades() {return EAD_Helicopter_Blades;}
     float EAD_Get_Helicopter_BladeRadius() {return EAD_Helicopter_BladeRadius;}
     float EAD_Get_Helicopter_BladeChord() {return EAD_Helicopter_BladeChord;}
     float EAD_Get_Helicopter_BladeC1() {return EAD_Helicopter_BladeC1;}
     float EAD_Get_Helicopter_BladeCD0() {return EAD_Helicopter_BladeCD0;}
     float EAD_Get_Helicopter_TipVelocity() {return EAD_Helicopter_TipVelocity;}
     float EAD_Get_Helicopter_FuselageArea() {return EAD_Helicopter_FuselageArea;}
     float EAD_Get_Helicopter_FuselageCd0() {return EAD_Helicopter_FuselageCd0;}

     // Set Methods from the parent
     void EAD_Set_Helicopter_ID(char* ID) {EAD_Aircraft::EAD_Set_Aircraft_ID(ID);}
     void EAD_Set_Helicopter_NonAerodynamic(int NonAerodynamic)
{EAD_Aircraft::EAD_Set_Aircraft_NonAerodynamic(NonAerodynamic);}
     void EAD_Set_Helicopter_MAXSpeed(int MAXSpeed)
{EAD_Aircraft::EAD_Set_Aircraft_MAXSpeed(MAXSpeed);}
     void EAD_Set_Helicopter_MINSpeed(int MINSpeed)
{EAD_Aircraft::EAD_Set_Aircraft_MINSpeed(MINSpeed);}
     void EAD_Set_Helicopter_MAXG(float MAXG)
{EAD_Aircraft::EAD_Set_Aircraft_MAXG(MAXG);}
     void EAD_Set_Helicopter_EmptyWeight(int EmptyWeight)
{EAD_Aircraft::EAD_Set_Aircraft_EmptyWeight(EmptyWeight);}
     void EAD_Set_Helicopter_FuelWeight(int FuelWeight)
{EAD_Aircraft::EAD_Set_Aircraft_FuelWeight(FuelWeight);}
     void EAD_Set_Helicopter_RTBFuelBingoLimit(int RTBFuelBingoLimit)
{EAD_Aircraft::EAD_Set_Aircraft_RTBFuelBingoLimit(RTBFuelBingoLimit);}
     void EAD_Set_Helicopter_AirRefuelBingoLimit(int AirRefuelBingoLimit)
{EAD_Aircraft::EAD_Set_Aircraft_AirRefuelBingoLimit(AirRefuelBingoLimit);}
     void EAD_Set_Helicopter_MaxFuelReceivingRate(int MaxFuelReceivingRate)
{EAD_Aircraft::EAD_Set_Aircraft_MaxFuelReceivingRate(MaxFuelReceivingRate);}
     void EAD_Set_Helicopter_LookAheadInterval(float LookAheadInterval)
{EAD_Aircraft::EAD_Set_Aircraft_LookAheadInterval(LookAheadInterval);}
     void EAD_Set_Helicopter_MultiplicationFactor(float MultiplicationFactor)
{EAD_Aircraft::EAD_Set_Aircraft_MultiplicationFactor(MultiplicationFactor);}
     void EAD_Set_Helicopter_TerrainSamples(int TerrainSamples)
{EAD_Aircraft::EAD_Set_Aircraft_TerrainSamples(TerrainSamples);}
     void EAD_Set_Helicopter_FeedbackControlGain(int FeedbackControlGain)
{EAD_Aircraft::EAD_Set_Aircraft_FeedbackControlGain(FeedbackControlGain);}
```

```
     void EAD_Set_Helicopter_MAXClimbAngle(int MAXClimbAngle)
{EAD_Aircraft::EAD_Set_Aircraft_MAXClimbAngle(MAXClimbAngle);}

     // Set Methods
     void EAD_Set_Helicopter_Power(float Power) {EAD_Helicopter_Power = Power;}
     void EAD_Set_Helicopter_PSFC(float PSFC) {EAD_Helicopter_PSFC = PSFC;}
     void EAD_Set_Helicopter_DecelarationG(float DecelarationG)
{EAD_Helicopter_DecelarationG = DecelarationG;}
     void EAD_Set_Helicopter_Blades(int Blades) {EAD_Helicopter_Blades = Blades;}
     void EAD_Set_Helicopter_BladeRadius(float BladeRadius) {EAD_Helicopter_BladeRadius =
BladeRadius;}
     void EAD_Set_Helicopter_BladeChord(float BladeChord) {EAD_Helicopter_BladeChord =
BladeChord;}
     void EAD_Set_Helicopter_BladeC1(float BladeC1) {EAD_Helicopter_BladeC1 = BladeC1;}
     void EAD_Set_Helicopter_BladeCD0(float BladeCD0) {EAD_Helicopter_BladeCD0 =
BladeCD0;}
     void EAD_Set_Helicopter_TipVelocity(float TipVelocity) {EAD_Helicopter_TipVelocity =
TipVelocity;}
     void EAD_Set_Helicopter_FuselageArea(float FuselageArea)
{EAD_Helicopter_FuselageArea = FuselageArea;}
     void EAD_Set_Helicopter_FuselageCd0(float FuselageCd0) {EAD_Helicopter_FuselageCd0 =
FuselageCd0;}

     // Constructor and Destructor Methods
     EAD_Helicopter(char* Airframe_ID, int NonAerodynamic, int MAXSpeed, int MINSpeed,
float MAXG, int EmptyWeight, int FuelWeight, int RTBFuelBingoLimit, int
AirRefuelBingoLimit, int MaxFuelReceivingRate, float LookAheadInterval, float
MultiplicationFactor, int TerrainSamples, int FeedbackControlGain, int MAXClimbAngle,
float Power, float PSFC, float DecelarationG, int Blades, float BladeRadius, float
BladeChord, float BladeC1, float BladeCD0, float TipVelocity, float FuselageArea, float
FuselageCd0);
     ~EAD_Helicopter();

     static os_typespec* get_os_typespec();

private:
     float EAD_Helicopter_Power;
     float EAD_Helicopter_PSFC;
     float EAD_Helicopter_DecelarationG;
     int EAD_Helicopter_Blades;
     float EAD_Helicopter_BladeRadius;
     float EAD_Helicopter_BladeChord;
     float EAD_Helicopter_BladeC1;
     float EAD_Helicopter_BladeCD0;
     float EAD_Helicopter_TipVelocity;
     float EAD_Helicopter_FuselageArea;
     float EAD_Helicopter_FuselageCd0;

};
```

Class EAD_Missile

EAD_Missile header

```
class EAD_Missile : EAD_Airframe
{
public:
     // Get Methods call parent
     char* EAD_Get_Missile_ID() {return EAD_Airframe::EAD_Get_Airframe_ID();}

     // Get Methods
     float EAD_Get_Missile_InitialVelocity() {return EAD_Missile_InitialVelocity;}
     float EAD_Get_Missile_LaunchRailLenght() {return EAD_Missile_LaunchRailLenght;}
     float EAD_Get_Missile_LaunchElevationAngle() {return
EAD_Missile_LaunchElevationAngle;}
```

```
     float EAD_Get_Missile_MAXDivertVelocity() {return EAD_Missile_MAXDivertVelocity;}
     float EAD_Get_Missile_MaxDivertLateralAcceleration() {return
EAD_Missile_MaxDivertLateralAcceleration;}
     os_Set<EAD_FlightSection*> EAD_Get_Missile_FlightSections() {return
EAD_Missile_FlightSections;}

      // Set Method from the parent
     void EAD_Set_Missile_ID(char* ID) {EAD_Airframe::EAD_Set_Airframe_ID(ID);}

     // Set Methods
     void EAD_Set_Missile_InitialVelocity(float InitialVelocity)
{EAD_Missile_InitialVelocity = InitialVelocity;}
     void EAD_Set_Missile_LaunchRailLenght(float LaunchRailLenght)
{EAD_Missile_LaunchRailLenght = LaunchRailLenght;}

     void EAD_Set_Missile_LaunchElevationAngle(float LaunchElevationAngle)
{EAD_Missile_LaunchElevationAngle = LaunchElevationAngle;}
     void EAD_Set_Missile_MAXDivertVelocity(float MAXDivertVelocity)
{EAD_Missile_MAXDivertVelocity = MAXDivertVelocity;}
     void EAD_Set_Missile_MaxDivertLateralAcceleration(float
MaxDivertLateralAcceleration) {EAD_Missile_MaxDivertLateralAcceleration =
MaxDivertLateralAcceleration;}
     void EAD_Set_Missile_FlightSection(EAD_FlightSection* FlighSection);

     // Constructor and Destructor Methods
     EAD_Missile(char* Airframe_ID, float InitialVelocity, float LaunchRailLenght, float
LaunchElevationAngle, float MAXDivertVelocity, float MaxDivertLateralAcceleration);
     ~EAD_Missile();
     static os_typespec* get_os_typespec();

private:
     float EAD_Missile_InitialVelocity;
     float EAD_Missile_LaunchRailLenght;
     float EAD_Missile_LaunchElevationAngle;
     float EAD_Missile_MAXDivertVelocity;
     float EAD_Missile_MaxDivertLateralAcceleration;
     os_Set<EAD_FlightSection*> EAD_Missile_FlightSections;
};
```

Class EAD_FlightSection

EAD_FlightSection header

```
class EAD_FlightSection
{
public:
      // Get Methods
     float EAD_Get_FlightSection_EndTime() {return EAD_FlightSection_EndTime;}
     float EAD_Get_FlightSection_MAXG() {return EAD_FlightSection_MAXG;}
     float EAD_Get_FlightSection_SpecificImpulse() {return
EAD_FlightSection_SpecificImpulse;}
     float EAD_Get_FlightSection_NozzleExitArea() {return
EAD_FlightSection_NozzleExitArea;}
     float EAD_Get_FlightSection_ReferenceArea() {return
EAD_FlightSection_ReferenceArea;}
     float EAD_Get_FlightSection_InitialMass() {return EAD_FlightSection_InitialMass;}
     float EAD_Get_FlightSection_MaxAlpha() {return EAD_FlightSection_MaxAlpha;}
     float EAD_Get_FlightSection_ResponseTime() {return EAD_FlightSection_ResponseTime;}
     float EAD_Get_FlightSection_ProNavGuidanceGain() {return
EAD_FlightSection_ProNavGuidanceGain;}
     float EAD_Get_FlightSection_IntegrationTimeStep() {return
EAD_FlightSection_IntegrationTimeStep;}
     float EAD_Get_FlightSection_IRSignature() {return EAD_FlightSection_IRSignature;}
     float EAD_Get_FlightSection_RCS() {return EAD_FlightSection_RCS;}
     // Set Methods
     void EAD_Set_FlightSection_EndTime(float EndTime) {EAD_FlightSection_EndTime =
EndTime;}
     void EAD_Set_FlightSection_MAXG(float MAXG) {EAD_FlightSection_MAXG = MAXG;}
     void EAD_Set_FlightSection_SpecificImpulse(float SpecificImpulse)
{EAD_FlightSection_SpecificImpulse = SpecificImpulse;}
     void EAD_Set_FlightSection_NozzleExitArea(float NozzleExitArea)
{EAD_FlightSection_NozzleExitArea = NozzleExitArea;}
     void EAD_Set_FlightSection_ReferenceArea(float ReferenceArea)
{EAD_FlightSection_ReferenceArea = ReferenceArea;}
     void EAD_Set_FlightSection_InitialMass(float InitialMass)
{EAD_FlightSection_InitialMass = InitialMass;}
     void EAD_Set_FlightSection_MaxAlpha(float MaxAlpha) {EAD_FlightSection_MaxAlpha =
MaxAlpha;}
     void EAD_Set_FlightSection_ResponseTime(float ResponseTime)
{EAD_FlightSection_ResponseTime = ResponseTime;}
     void EAD_Set_FlightSection_ProNavGuidanceGain(float ProNavGuidanceGain)
{EAD_FlightSection_ProNavGuidanceGain = ProNavGuidanceGain;}
     void EAD_Set_FlightSection_IntegrationTimeStep(float IntegrationTimeStep)
{EAD_FlightSection_IntegrationTimeStep = IntegrationTimeStep;}
     void EAD_Set_FlightSection_IRSignature(float IRSignature)
{EAD_FlightSection_IRSignature = IRSignature;}
     void EAD_Set_FlightSection_RCS(float RCS) {EAD_FlightSection_RCS = RCS;}

     // Constructor and Destructor Methods
     EAD_FlightSection(float EndTime, float MAXG, float SpecificImpulse, float
NozzleExitArea, float ReferenceArea, float InitialMass, float MaxAlpha, float
ResponseTime, float ProNavGuidanceGain, float IntegrationTimeStep, float IRSignature,
float RCS);
     ~EAD_FlightSection();
     static os_typespec* get_os_typespec();

private:
     float EAD_FlightSection_EndTime;
     float EAD_FlightSection_MAXG;
     float EAD_FlightSection_SpecificImpulse;
     float EAD_FlightSection_NozzleExitArea;
     float EAD_FlightSection_ReferenceArea;
     float EAD_FlightSection_InitialMass;
     float EAD_FlightSection_MaxAlpha;
     float EAD_FlightSection_ResponseTime;
     float EAD_FlightSection_ProNavGuidanceGain;
     float EAD_FlightSection_IntegrationTimeStep;
     float EAD_FlightSection_IRSignature;
     float EAD_FlightSection_RCS;
};
```

5.2.3. Integrating metadata

The Integration Dictionary system was used to insert metadata into the database. Since the global schema was empty, there was no schema conflict to solve.

5.2.4. Integrating schema

After the metadata of the EADSIM exported schema was integrated into the global schema, the schema code was also integrated. Since the global schema was empty, the schema code for the global schema was written and the programs to update the schema structure or to perform data migration from the old schema to the new schema didn't need to be written. The global schema object model is the same as shown in Figure 18. The body codes of the global classes' schema are shown in Appendix C, Section 1. Below I present the header code of the global classes' schema.

Airframe

Airframe header

```
class Airframe
{
public:
     // Get Methods
     char* Get_Airframe_ID() {return Airframe_ID;}

     //Set Methods
     void Set_Airframe_ID(const char* ID) {strcpy(Airframe_ID, ID);}

     // Show method
     void Airframe_Show(osstream& os);

     // Constructor and Destructor Methods
     Airframe(const char* ID);
     ~Airframe();

     static os_typespec* get_os_typespec();

protected:
     char Airframe_ID[31];
};
```

Aircraft

Aircraft header

```
class Aircraft : Airframe
{
public:
     // Get Method call parent
     char* Get_Aircraft_ID() {return Airframe::Get_Airframe_ID();}


     // Get Methods
     int Get_Aircraft_NonAerodynamic() {return Aircraft_NonAerodynamic;}
     int Get_Aircraft_MAXSpeed() {return Aircraft_MAXSpeed;}
     int Get_Aircraft_MINSpeed() {return Aircraft_MINSpeed;}
     float Get_Aircraft_MAXG() {return Aircraft_MAXG;}
     int Get_Aircraft_EmptyWeight() {return Aircraft_EmptyWeight;}
     int Get_Aircraft_FuelWeight() {return Aircraft_FuelWeight;}
     int Get_Aircraft_RTBFuelBingoLimit() {return Aircraft_RTBFuelBingoLimit;}
     int Get_Aircraft_AirRefuelBingoLimit() {return Aircraft_AirRefuelBingoLimit;}
     int Get_Aircraft_MaxFuelReceivingRate() {return Aircraft_MaxFuelReceivingRate;}
     float Get_Aircraft_LookAheadInterval() {return Aircraft_LookAheadInterval;}
     float Get_Aircraft_MultiplicationFactor() {return Aircraft_MultiplicationFactor;}
     int Get_Aircraft_TerrainSamples() {return Aircraft_TerrainSamples;}
     int Get_Aircraft_FeedbackControlGain() {return Aircraft_FeedbackControlGain;}
     int Get_Aircraft_MAXClimbAngle() {return Aircraft_MAXClimbAngle;}

     // Set Methods from the parent
     void Set_Aircraft_ID(char* ID) {Airframe::Set_Airframe_ID(ID);}

     // Set Methods
     void Set_Aircraft_NonAerodynamic(int NonAerodynamic) {Aircraft_NonAerodynamic =
NonAerodynamic;}
     void Set_Aircraft_MAXSpeed(int MAXSpeed) {Aircraft_MAXSpeed = MAXSpeed;}
     void Set_Aircraft_MINSpeed(int MINSpeed) {Aircraft_MINSpeed = MINSpeed;}
     void Set_Aircraft_MAXG(float MAXG) {Aircraft_MAXG = MAXG;}
     void Set_Aircraft_EmptyWeight(int EmptyWeight) {Aircraft_EmptyWeight = EmptyWeight;}
     void Set_Aircraft_FuelWeight(int FuelWeight) {Aircraft_FuelWeight = FuelWeight;}
     void Set_Aircraft_RTBFuelBingoLimit(int RTBFuelBingoLimit)
{Aircraft_RTBFuelBingoLimit = RTBFuelBingoLimit;}
     void Set_Aircraft_AirRefuelBingoLimit(int AirRefuelBingoLimit)
{Aircraft_AirRefuelBingoLimit = AirRefuelBingoLimit;}
     void Set_Aircraft_MaxFuelReceivingRate(int MaxFuelReceivingRate)
{Aircraft_MaxFuelReceivingRate = MaxFuelReceivingRate;}
     void Set_Aircraft_LookAheadInterval(float LookAheadInterval)
{Aircraft_LookAheadInterval = LookAheadInterval;}
     void Set_Aircraft_MultiplicationFactor(float MultiplicationFactor)
{Aircraft_MultiplicationFactor = MultiplicationFactor;}
     void Set_Aircraft_TerrainSamples(int TerrainSamples) {Aircraft_TerrainSamples =
TerrainSamples;}
     void Set_Aircraft_FeedbackControlGain(int FeedbackControlGain)
{Aircraft_FeedbackControlGain = FeedbackControlGain;}
     void Set_Aircraft_MAXClimbAngle(int MAXClimbAngle) {Aircraft_MAXClimbAngle =
MAXClimbAngle;}

     // Constructor and Destructor Methods
     Aircraft(char* Airframe_ID, int NonAerodynamic, int MAXSpeed, int MINSpeed, float
MAXG, int EmptyWeight, int FuelWeight, int RTBFuelBingoLimit, int AirRefuelBingoLimit,
int MaxFuelReceivingRate, float LookAheadInterval, float MultiplicationFactor, int
TerrainSamples, int FeedbackControlGain, int MAXClimbAngle);
     ~Aircraft();

     // Show method
     void Aircraft_Show(osstream& os);

     static os_typespec* get_os_typespec();
protected:
     int Aircraft_NonAerodynamic;
     int Aircraft_MAXSpeed;
     int Aircraft_MINSpeed;
     float Aircraft_MAXG;
     int Aircraft_EmptyWeight;
     int Aircraft_FuelWeight;
     int Aircraft_RTBFuelBingoLimit;
     int Aircraft_AirRefuelBingoLimit;
     int Aircraft_MaxFuelReceivingRate;
     float Aircraft_LookAheadInterval;
     float Aircraft_MultiplicationFactor;
     int Aircraft_TerrainSamples;
     int Aircraft_FeedbackControlGain;
     int Aircraft_MAXClimbAngle;
};
```

Airplane

Airplane header

```
class Airplane : Aircraft
{
public:
     // Get Methods call parent
     char* Get_Airplane_ID() {return Aircraft::Get_Aircraft_ID();}
     int Get_Airplane_NonAerodynamic() {return Aircraft::Get_Aircraft_NonAerodynamic();}
     int Get_Airplane_MAXSpeed() {return Aircraft::Get_Aircraft_MAXSpeed();}
     int Get_Airplane_MINSpeed() {return Aircraft::Get_Aircraft_MINSpeed();}
     float Get_Airplane_MAXG() {return Aircraft::Get_Aircraft_MAXG();}
     int Get_Airplane_EmptyWeight() {return Aircraft::Get_Aircraft_EmptyWeight();}
     int Get_Airplane_FuelWeight() {return Aircraft::Get_Aircraft_FuelWeight();}
     int Get_Airplane_RTBFuelBingoLimit() {return
Aircraft::Get_Aircraft_RTBFuelBingoLimit();}
     int Get_Airplane_AirRefuelBingoLimit() {return
Aircraft::Get_Aircraft_AirRefuelBingoLimit();}
     int Get_Airplane_MaxFuelReceivingRate() {return
Aircraft::Get_Aircraft_MaxFuelReceivingRate();}

     float Get_Airplane_LookAheadInterval() {return
Aircraft::Get_Aircraft_LookAheadInterval();}
     float Get_Airplane_MultiplicationFactor() {return
Aircraft::Get_Aircraft_MultiplicationFactor();}
     int Get_Airplane_TerrainSamples() {return Aircraft::Get_Aircraft_TerrainSamples();}
     int Get_Airplane_FeedbackControlGain() {return
Aircraft::Get_Aircraft_FeedbackControlGain();}
     int Get_Airplane_MAXClimbAngle() {return Aircraft::Get_Aircraft_MAXClimbAngle();}

     // Get Methods
     int Get_Airplane_ABSpeed() {return Airplane_ABSpeed;}
     int Get_Airplane_MAXThrust() {return Airplane_MAXThrust;}
     float Get_Airplane_WingArea() {return Airplane_WingArea;}
     float Get_Airplane_WingSpan() {return Airplane_WingSpan;}
     int Get_Airplane_CruiseAltMSL() {return Airplane_CruiseAltMSL;}
     int Get_Airplane_CruiseSpeed() {return Airplane_CruiseSpeed;}
     int Get_Airplane_FuelFlow() {return Airplane_FuelFlow;}
     float Get_Airplane_TSFC() {return Airplane_TSFC;}
     int Get_Airplane_MAXSpeedCruiseAlt() {return Airplane_MAXSpeedCruiseAlt;}
     float Get_Airplane_CD0() {return Airplane_CD0;}

     // Set Methods from the parent
     void Set_Airplane_ID(char* ID) {Aircraft::Set_Aircraft_ID(ID);}
     void Set_Airplane_NonAerodynamic(int NonAerodynamic)
{Aircraft::Set_Aircraft_NonAerodynamic(NonAerodynamic);}
     void Set_Airplane_MAXSpeed(int MAXSpeed)
{Aircraft::Set_Aircraft_MAXSpeed(MAXSpeed);}
     void Set_Airplane_MINSpeed(int MINSpeed)
{Aircraft::Set_Aircraft_MINSpeed(MINSpeed);}
     void Set_Airplane_MAXG(float MAXG) {Aircraft::Set_Aircraft_MAXG(MAXG);}
     void Set_Airplane_EmptyWeight(int EmptyWeight)
{Aircraft::Set_Aircraft_EmptyWeight(EmptyWeight);}
     void Set_Airplane_FuelWeight(int FuelWeight)
{Aircraft::Set_Aircraft_FuelWeight(FuelWeight);}
     void Set_Airplane_RTBFuelBingoLimit(int RTBFuelBingoLimit)
{Aircraft::Set_Aircraft_RTBFuelBingoLimit(RTBFuelBingoLimit);}
     void Set_Airplane_AirRefuelBingoLimit(int AirRefuelBingoLimit)
{Aircraft::Set_Aircraft_AirRefuelBingoLimit(AirRefuelBingoLimit);}
     void Set_Airplane_MaxFuelReceivingRate(int MaxFuelReceivingRate)
{Aircraft::Set_Aircraft_MaxFuelReceivingRate(MaxFuelReceivingRate);}
     void Set_Airplane_LookAheadInterval(float LookAheadInterval)
{Aircraft::Set_Aircraft_LookAheadInterval(LookAheadInterval);}
     void Set_Airplane_MultiplicationFactor(float MultiplicationFactor)
{Aircraft::Set_Aircraft_MultiplicationFactor(MultiplicationFactor);}
     void Set_Airplane_TerrainSamples(int TerrainSamples)
{Aircraft::Set_Aircraft_TerrainSamples(TerrainSamples);}
     void Set_Airplane_FeedbackControlGain(int FeedbackControlGain)
{Aircraft::Set_Aircraft_FeedbackControlGain(FeedbackControlGain);}
```

```
     void Set_Airplane_MAXClimbAngle(int MAXClimbAngle)
{Aircraft::Set_Aircraft_MAXClimbAngle(MAXClimbAngle);}

     // Set Methods
     void Set_Airplane_ABSpeed(int ABSpeed) {Airplane_ABSpeed = ABSpeed;}
     void Set_Airplane_MAXThrust(int MAXThrust) {Airplane_MAXThrust = MAXThrust;}
     void Set_Airplane_WingArea(float WingArea) {Airplane_WingArea = WingArea;}
     void Set_Airplane_WingSpan(float WingSpan) {Airplane_WingSpan = WingSpan;}
     void Set_Airplane_CruiseAltMSL(int CruiseAltMSL) {Airplane_CruiseAltMSL =
CruiseAltMSL;}
     void Set_Airplane_CruiseSpeed(int CruiseSpeed) {Airplane_CruiseSpeed = CruiseSpeed;}
     void Set_Airplane_FuelFlow(int FuelFlow) {Airplane_FuelFlow = FuelFlow;}
     void Set_Airplane_TSFC(float TSFC) {Airplane_TSFC = TSFC;}
     void Set_Airplane_MAXSpeedCruiseAlt(int MAXSpeedCruiseAlt)
{Airplane_MAXSpeedCruiseAlt = MAXSpeedCruiseAlt;}
     void Set_Airplane_CD0(float CD0) {Airplane_CD0 = CD0;}

     // Show method
     void Airplane_Show(osstream& os);

     // Constructor and Destructor Methods
     Airplane(char* Airframe_ID, int NonAerodynamic, int MAXSpeed, int MINSpeed, float
MAXG, int EmptyWeight, int FuelWeight, int RTBFuelBingoLimit, int AirRefuelBingoLimit,

int MaxFuelReceivingRate, float LookAheadInterval, float MultiplicationFactor, int
TerrainSamples, int FeedbackControlGain, int MAXClimbAngle, int ABSpeed, int MAXThrust,
float WingArea, float WingSpan, int CruiseAltMSL, int CruiseSpeed, int FuelFlow, float
TSFC, int MAXSpeedCruiseAlt, float CD0);
     ~Airplane();

     static os_typespec* get_os_typespec();

private:
     int Airplane_ABSpeed;
     int Airplane_MAXThrust;
     float Airplane_WingArea;
     float Airplane_WingSpan;
     int Airplane_CruiseAltMSL;
     int Airplane_CruiseSpeed;
     int Airplane_FuelFlow;
     float Airplane_TSFC;
     int Airplane_MAXSpeedCruiseAlt;
     float Airplane_CD0;
};
```

Helicopter

Helicopter header

```
class Helicopter : Aircraft
{
public:
        // Get Methods call parent
     char* Get_Helicopter_ID() {return Aircraft::Get_Aircraft_ID();}
        int Get_Helicopter_NonAerodynamic() {return
Aircraft::Get_Aircraft_NonAerodynamic();}
     int Get_Helicopter_MAXSpeed() {return Aircraft::Get_Aircraft_MAXSpeed();}
     int Get_Helicopter_MINSpeed() {return Aircraft::Get_Aircraft_MINSpeed();}
     float Get_Helicopter_MAXG() {return Aircraft::Get_Aircraft_MAXG();}
     int Get_Helicopter_EmptyWeight() {return Aircraft::Get_Aircraft_EmptyWeight();}
     int Get_Helicopter_FuelWeight() {return Aircraft::Get_Aircraft_FuelWeight();}
     int Get_Helicopter_RTBFuelBingoLimit() {return
Aircraft::Get_Aircraft_RTBFuelBingoLimit();}
     int Get_Helicopter_AirRefuelBingoLimit() {return
Aircraft::Get_Aircraft_AirRefuelBingoLimit();}
```

```
     int Get_Helicopter_MaxFuelReceivingRate() {return
Aircraft::Get_Aircraft_MaxFuelReceivingRate();}
     float Get_Helicopter_LookAheadInterval() {return
Aircraft::Get_Aircraft_LookAheadInterval();}
     float Get_Helicopter_MultiplicationFactor() {return
Aircraft::Get_Aircraft_MultiplicationFactor();}
     int Get_Helicopter_TerrainSamples() {return
Aircraft::Get_Aircraft_TerrainSamples();}
     int Get_Helicopter_FeedbackControlGain() {return
Aircraft::Get_Aircraft_FeedbackControlGain();}
     int Get_Helicopter_MAXClimbAngle() {return Aircraft::Get_Aircraft_MAXClimbAngle();}

     // Get Methods
     float Get_Helicopter_Power() {return Helicopter_Power;}
     float Get_Helicopter_PSFC() {return Helicopter_PSFC;}
     float Get_Helicopter_DecelarationG() {return Helicopter_DecelarationG;}
     int Get_Helicopter_Blades() {return Helicopter_Blades;}
     float Get_Helicopter_BladeRadius() {return Helicopter_BladeRadius;}
     float Get_Helicopter_BladeChord() {return Helicopter_BladeChord;}
     float Get_Helicopter_BladeCl() {return Helicopter_BladeCl;}
     float Get_Helicopter_BladeCD0() {return Helicopter_BladeCD0;}
     float Get_Helicopter_TipVelocity() {return Helicopter_TipVelocity;}
     float Get_Helicopter_FuselageArea() {return Helicopter_FuselageArea;}
     float Get_Helicopter_FuselageCd0() {return Helicopter_FuselageCd0;}

     // Set Methods from the parent
     void Set_Helicopter_ID(char* ID) {Aircraft::Set_Aircraft_ID(ID);}
     void Set_Helicopter_NonAerodynamic(int NonAerodynamic)
{Aircraft::Set_Aircraft_NonAerodynamic(NonAerodynamic);}
     void Set_Helicopter_MAXSpeed(int MAXSpeed)
{Aircraft::Set_Aircraft_MAXSpeed(MAXSpeed);}
     void Set_Helicopter_MINSpeed(int MINSpeed)
{Aircraft::Set_Aircraft_MINSpeed(MINSpeed);}
     void Set_Helicopter_MAXG(float MAXG) {Aircraft::Set_Aircraft_MAXG(MAXG);}
     void Set_Helicopter_EmptyWeight(int EmptyWeight)
{Aircraft::Set_Aircraft_EmptyWeight(EmptyWeight);}
     void Set_Helicopter_FuelWeight(int FuelWeight)
{Aircraft::Set_Aircraft_FuelWeight(FuelWeight);}
     void Set_Helicopter_RTBFuelBingoLimit(int RTBFuelBingoLimit)
{Aircraft::Set_Aircraft_RTBFuelBingoLimit(RTBFuelBingoLimit);}
     void Set_Helicopter_AirRefuelBingoLimit(int AirRefuelBingoLimit)
{Aircraft::Set_Aircraft_AirRefuelBingoLimit(AirRefuelBingoLimit);}
     void Set_Helicopter_MaxFuelReceivingRate(int MaxFuelReceivingRate)
{Aircraft::Set_Aircraft_MaxFuelReceivingRate(MaxFuelReceivingRate);}
     void Set_Helicopter_LookAheadInterval(float LookAheadInterval)
{Aircraft::Set_Aircraft_LookAheadInterval(LookAheadInterval);}
     void Set_Helicopter_MultiplicationFactor(float MultiplicationFactor)
{Aircraft::Set_Aircraft_MultiplicationFactor(MultiplicationFactor);}
     void Set_Helicopter_TerrainSamples(int TerrainSamples)
{Aircraft::Set_Aircraft_TerrainSamples(TerrainSamples);}
     void Set_Helicopter_FeedbackControlGain(int FeedbackControlGain)
{Aircraft::Set_Aircraft_FeedbackControlGain(FeedbackControlGain);}
     void Set_Helicopter_MAXClimbAngle(int MAXClimbAngle)
{Aircraft::Set_Aircraft_MAXClimbAngle(MAXClimbAngle);}

     // Set Methods
     void Set_Helicopter_Power(float Power) {Helicopter_Power = Power;}
     void Set_Helicopter_PSFC(float PSFC) {Helicopter_PSFC = PSFC;}
     void Set_Helicopter_DecelarationG(float DecelarationG) {Helicopter_DecelarationG =
DecelarationG;}
     void Set_Helicopter_Blades(int Blades) {Helicopter_Blades = Blades;}
     void Set_Helicopter_BladeRadius(float BladeRadius) {Helicopter_BladeRadius =
BladeRadius;}
     void Set_Helicopter_BladeChord(float BladeChord) {Helicopter_BladeChord =
BladeChord;}
     void Set_Helicopter_BladeCl(float BladeCl) {Helicopter_BladeCl = BladeCl;}
     void Set_Helicopter_BladeCD0(float BladeCD0) {Helicopter_BladeCD0 = BladeCD0;}
     void Set_Helicopter_TipVelocity(float TipVelocity) {Helicopter_TipVelocity =
TipVelocity;}
     void Set_Helicopter_FuselageArea(float FuselageArea) {Helicopter_FuselageArea =
FuselageArea;}
```

```
     void Set_Helicopter_FuselageCd0(float FuselageCd0) {Helicopter_FuselageCd0 =
FuselageCd0;}

     // Show method
     void Helicopter_Show(osstream& os);

     // Constructor and Destructor Methods
     Helicopter(char* Airframe_ID, int NonAerodynamic, int MAXSpeed, int MINSpeed, float
MAXG, int EmptyWeight, int FuelWeight, int RTBFuelBingoLimit, int AirRefuelBingoLimit,
int MaxFuelReceivingRate, float LookAheadInterval, float MultiplicationFactor, int
TerrainSamples, int FeedbackControlGain, int MAXClimbAngle, float Power, float PSFC,
float DecelarationG, int Blades, float BladeRadius, float BladeChord, float BladeC1,
float BladeCD0, float TipVelocity, float FuselageArea, float FuselageCd0);
     ~Helicopter();

     static os_typespec* get_os_typespec();

private:
     float Helicopter_Power;
     float Helicopter_PSFC;
     float Helicopter_DecelarationG;
     int Helicopter_Blades;
     float Helicopter_BladeRadius;
     float Helicopter_BladeChord;
     float Helicopter_BladeC1;
     float Helicopter_BladeCD0;
     float Helicopter_TipVelocity;
     float Helicopter_FuselageArea;
     float Helicopter_FuselageCd0;
};
```

Missile

Missile header

```
class Missile : Airframe
{
public:
         // Get Method call parent
     char* Get_Missile_ID() {return Airframe::Get_Airframe_ID();}

     // Get Methods
     float Get_Missile_InitialVelocity() {return Missile_InitialVelocity;}
     float Get_Missile_LaunchRailLenght() {return Missile_LaunchRailLenght;}
     float Get_Missile_LaunchElevationAngle() {return Missile_LaunchElevationAngle;}
     float Get_Missile_MAXDivertVelocity() {return Missile_MAXDivertVelocity;}
     float Get_Missile_MaxDivertLateralAcceleration() {return
Missile_MaxDivertLateralAcceleration;}
     os_Set<FlightSection*> Get_Missile_FlightSections() {return Missile_FlightSections;}

     // Set Methods from the parent
     void Set_Missile_ID(char* ID) {Airframe::Set_Airframe_ID(ID);}

     // Set Methods
     void Set_Missile_InitialVelocity(float InitialVelocity) {Missile_InitialVelocity =
InitialVelocity;}
     void Set_Missile_LaunchRailLenght(float LaunchRailLenght) {Missile_LaunchRailLenght
= LaunchRailLenght;}
     void Set_Missile_LaunchElevationAngle(float LaunchElevationAngle)
{Missile_LaunchElevationAngle = LaunchElevationAngle;}
     void Set_Missile_MAXDivertVelocity(float MAXDivertVelocity)
{Missile_MAXDivertVelocity = MAXDivertVelocity;}

     void Set_Missile_MaxDivertLateralAcceleration(float MaxDivertLateralAcceleration)
{Missile_MaxDivertLateralAcceleration = MaxDivertLateralAcceleration;}
```

```
     void Set_Missile_FlightSection(FlightSection* FlighSection);


     // Show method
     void Missile_Show(osstream& os);

     // Constructor and Destructor Methods
     Missile(char* Airframe_ID, float InitialVelocity, float LaunchRailLenght, float
LaunchElevationAngle, float MAXDivertVelocity, float MaxDivertLateralAcceleration);
     ~Missile();

     static os_typespec* get_os_typespec();

private:
     float Missile_InitialVelocity;
     float Missile_LaunchRailLenght;
     float Missile_LaunchElevationAngle;
     float Missile_MAXDivertVelocity;
     float Missile_MaxDivertLateralAcceleration;
     os_Set<FlightSection*> Missile_FlightSections;
};
```

FlightSection

FlightSection header

```
class FlightSection
{
public:
     // Get Methods
     float Get_FlightSection_EndTime() {return FlightSection_EndTime;}
     float Get_FlightSection_MAXG() {return FlightSection_MAXG;}
     float Get_FlightSection_SpecificImpulse() {return FlightSection_SpecificImpulse;}
     float Get_FlightSection_NozzleExitArea() {return FlightSection_NozzleExitArea;}
     float Get_FlightSection_ReferenceArea() {return FlightSection_ReferenceArea;}
     float Get_FlightSection_InitialMass() {return FlightSection_InitialMass;}
     float Get_FlightSection_MaxAlpha() {return FlightSection_MaxAlpha;}
     float Get_FlightSection_ResponseTime() {return FlightSection_ResponseTime;}
     float Get_FlightSection_ProNavGuidanceGain() {return
FlightSection_ProNavGuidanceGain;}
     float Get_FlightSection_IntegrationTimeStep() {return
FlightSection_IntegrationTimeStep;}
     float Get_FlightSection_IRSignature() {return FlightSection_IRSignature;}
     float Get_FlightSection_RCS() {return FlightSection_RCS;}

     // Set Methods
     void Set_FlightSection_EndTime(float EndTime) {FlightSection_EndTime = EndTime;}
     void Set_FlightSection_MAXG(float MAXG) {FlightSection_MAXG = MAXG;}
     void Set_FlightSection_SpecificImpulse(float SpecificImpulse)
{FlightSection_SpecificImpulse = SpecificImpulse;}
     void Set_FlightSection_NozzleExitArea(float NozzleExitArea)
{FlightSection_NozzleExitArea = NozzleExitArea;}
     void Set_FlightSection_ReferenceArea(float ReferenceArea)
{FlightSection_ReferenceArea = ReferenceArea;}
     void Set_FlightSection_InitialMass(float InitialMass) {FlightSection_InitialMass =
InitialMass;}
     void Set_FlightSection_MaxAlpha(float MaxAlpha) {FlightSection_MaxAlpha = MaxAlpha;}
     void Set_FlightSection_ResponseTime(float ResponseTime) {FlightSection_ResponseTime
= ResponseTime;}
     void Set_FlightSection_ProNavGuidanceGain(float ProNavGuidanceGain)
{FlightSection_ProNavGuidanceGain = ProNavGuidanceGain;}
     void Set_FlightSection_IntegrationTimeStep(float IntegrationTimeStep)
{FlightSection_IntegrationTimeStep = IntegrationTimeStep;}
     void Set_FlightSection_IRSignature(float IRSignature) {FlightSection_IRSignature =
IRSignature;}
```

```
     void Set_FlightSection_RCS(float RCS) {FlightSection_RCS = RCS;}

     // Show method
     void FlightSection_Show(osstream& os);

     // Constructor and Destructor Methods
     FlightSection(float EndTime, float MAXG, float SpecificImpulse, float
NozzleExitArea, float ReferenceArea, float InitialMass, float MaxAlpha, float
ResponseTime, float ProNavGuidanceGain, float IntegrationTimeStep, float IRSignature,
float RCS);
     ~FlightSection();

     static os_typespec* get_os_typespec();
private:
     float FlightSection_EndTime;
     float FlightSection_MAXG;
     float FlightSection_SpecificImpulse;
     float FlightSection_NozzleExitArea;
     float FlightSection_ReferenceArea;
     float FlightSection_InitialMass;
     float FlightSection_MaxAlpha;
     float FlightSection_ResponseTime;
     float FlightSection_ProNavGuidanceGain;
     float FlightSection_IntegrationTimeStep;
     float FlightSection_IRSignature;
     float FlightSection_RCS;
};
```

5.2.5. Integrating data

Data from the exported schema must be merged with the data constant of the global schema. Since the global schema was empty, there was no data conflict to be solved. A specific program to convert the data from the exported schema to the global schema was written manually.

5.2.6. User's View definition

The User's View Construct program is used to help the IA with the User's View generation. The User's View translation program for EADSIM was written manually based on the output generated by the User's View Construct program listed in Appendix A, Section 1.

5.3. *Integrating Suppressor*

The following sub-sections of this section describe all the steps used to integrate the Suppressor Scenario Database (OO-DBMS) into the global database.

5.3.1. *Scope for integration*

Part of the Suppressor scenario database schema was selected to demonstrate the integration. In Figure 19 the system Object Model is shown. A Player aggregates one or more Locations. A Location aggregates one or more Elements. An Element aggregates one or more Systems. A System aggregates basic resources, which can be Radar, Sensor, Weapon, Mover, Communication, Jammer etc. Each resource has capabilities.

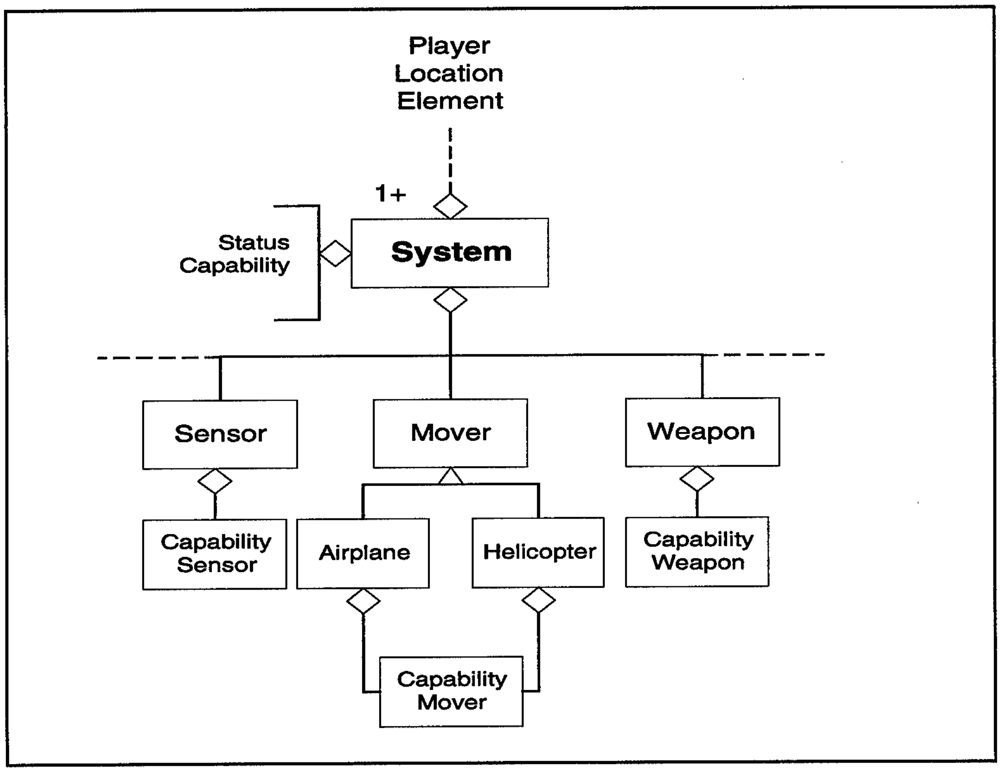


Figure 19: Suppressor Object Model

In order to demonstrate the methodology for database integration, the Mover resource hierarchy and its aggregations were chosen to be implemented in the global schema example. The scope for the implementation is shown in Figure 20.

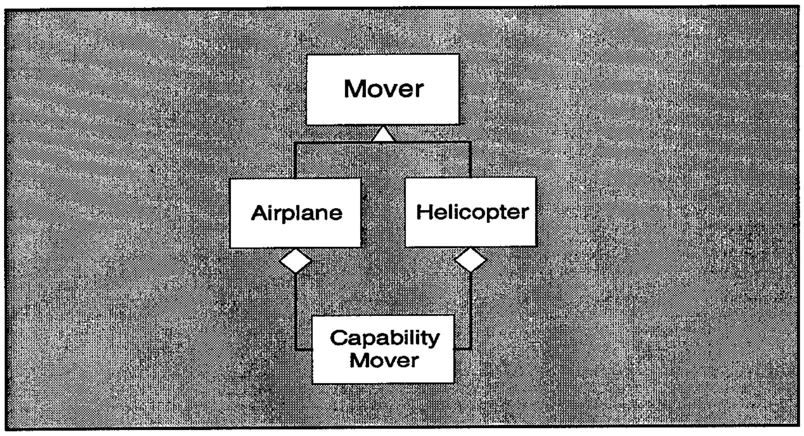


Figure 20: Object Model scope for implementation

5.3.2. Definition of the exported schema

For each class shown in Figure 20, the implementation code was created using the Object-Store definition language. According to Section 4.7 of Chapter 4, the naming rule must be applied to allow the User's View generation. All schema components have SUP in the three first characters of their names.

Also, only the lower level classes of the hierarchy (MoverAirplane and MoverHelicopter) were implemented as concrete classes. The body codes of the Suppressor exported classes' schema are shown in Appendix B, Section 2. Below I present the header code of the classes' schema.

SUP_Mover

SUP_Mover header

```
class SUP_Mover
{
public:
     // Get Methods
     int SUP_Get_Mover_SystemID() {return SUP_Mover_SystemID;}
     char* SUP_Get_Mover_Name() {return SUP_Mover_Name;}
     char* SUP_Get_Mover_Type() {return SUP_Mover_Type;}

     // Set Methods
     void SUP_Set_Mover_SystemID(int SystemID) {SUP_Mover_SystemID = SystemID;}
     void SUP_Set_Mover_Name(char* Name) {strcpy(SUP_Mover_Name, Name);
     void SUP_Set_Mover_Type(char* MType) {strcpy(SUP_Mover_Type, MType);}

protected:
     // Constructor and Destructor Methods
     SUP_Mover(int SystemID, const char* Name);
     ~SUP_Mover();

     int SUP_Mover_SystemID;
     char SUP_Mover_Name[31];
     char SUP_Mover_Type[51];
};
```

SUP_MoverAirplane

SUP_MoverAirplane header

```
class SUP_MoverAirplane : SUP_Mover

{
public:
  // Constructor and Destructor Methods
  SUP_MoverAirplane(int SystemID, const char* MoverName, SUP_CapabilityMover* CapMov);
  ~SUP_MoverAirplane();

  // Get Methods from parent
  int SUP_Get_MoverAirplane_SystemID() {return SUP_Mover::SUP_Get_Mover_SystemID();}
  char* SUP_Get_MoverAirplane_Name() {return SUP_Mover::SUP_Get_Mover_Name();}
  char* SUP_Get_MoverAirplane_Type() {return SUP_Mover::SUP_Get_Mover_Type();}
  SUP_CapabilityMover* SUP_Get_MoverAirplane_CapabilityAirplane() {return
SUP_MoverAirplane_CapabilityAirplane;}

  // Set Methods from parent
  void SUP_Set_MoverAirplane_SystemID(int SystemID)
{SUP_Mover::SUP_Set_Mover_SystemID(SystemID);}
  void SUP_Set_MoverAirplane_Name(char* Name) {SUP_Mover::SUP_Set_Mover_Name(Name);}
  void SUP_Set_MoverAirplane_Type(char* MType) {SUP_Mover::SUP_Set_Mover_Type(MType);}
  void SUP_Set_MoverAirplane_CapabilityAirplane(SUP_CapabilityMover* CapMov)
{SUP_MoverAirplane_CapabilityAirplane = CapMov;}

  static os_typespec* get_os_typespec();

private:
  SUP_CapabilityMover* SUP_MoverAirplane_CapabilityAirplane;
};
```

SUP_MoverHelicopter

SUP_MoverHelicopter header

```
class SUP_MoverHelicopter : SUP_Mover

{
public:
  // Constructor and Destructor Methods
  SUP_MoverHelicopter(int SystemID, const char* MoverName, SUP_CapabilityMover* CapMov);
  ~SUP_MoverHelicopter();

  // Get Methods from parent
  int SUP_Get_MoverHelicopter_SystemID() {return SUP_Mover::SUP_Get_Mover_SystemID();}
  char* SUP_Get_MoverHelicopter_Name() {return SUP_Mover::SUP_Get_Mover_Name();}
  char* SUP_Get_MoverHelicopter_Type() {return SUP_Mover::SUP_Get_Mover_Type();}
  SUP_CapabilityMover* SUP_Get_MoverHelicopter_CapabilityHelicopter() {return
SUP_MoverHelicopter_CapabilityHelicopter;}

  // Set Methods from parent
  void SUP_Set_MoverHelicopter_SystemID(int SystemID)
{SUP_Mover::SUP_Set_Mover_SystemID(SystemID);}
  void SUP_Set_MoverHelicopter_Name(char* Name) {SUP_Mover::SUP_Set_Mover_Name(Name);}
  void SUP_Set_MoverHelicopter_Type(char* MType) {SUP_Mover::SUP_Set_Mover_Type(MType);}
  void SUP_Set_MoverHelicopter_CapabilityHelicopter(SUP_CapabilityMover* CapMov)
{SUP_MoverHelicopter_CapabilityHelicopter = CapMov;}

  static os_typespec* get_os_typespec();

private:
  SUP_CapabilityMover* SUP_MoverHelicopter_CapabilityHelicopter;
};
```

SUP_CapabilityMover

SUP_CapabilityMover header

```
class SUP_CapabilityMover
{
public:
     // Get Methods
     const char* SUP_Get_CapabilityMover_Name() {return SUP_CapabilityMover_Name;}
     const char* SUP_Get_CapabilityMover_CommitAlt() {return
SUP_CapabilityMover_CommitAlt;}
     const char* SUP_Get_CapabilityMover_FuelUsage() {return
SUP_CapabilityMover_FuelUsage;}
     const char* SUP_Get_CapabilityMover_NavErrorData() {return
SUP_CapabilityMover_NavErrorData;}
     float SUP_Get_CapabilityMover_MAXAcceleration() {return
SUP_CapabilityMover_MAXAcceleration;}
     float SUP_Get_CapabilityMover_MINSpeed() {return SUP_CapabilityMover_MINSpeed;}
     float SUP_Get_CapabilityMover_MAXSpeed() {return SUP_CapabilityMover_MAXSpeed;}
     float SUP_Get_CapabilityMover_MINTurnRadius() {return
SUP_CapabilityMover_MINTurnRadius;}
     float SUP_Get_CapabilityMover_MINAltitude() {return
SUP_CapabilityMover_MINAltitude;}
     float SUP_Get_CapabilityMover_MAXAltitude() {return
SUP_CapabilityMover_MAXAltitude;}
     float SUP_Get_CapabilityMover_MAXDiveRate() {return
SUP_CapabilityMover_MAXDiveRate;}
     float SUP_Get_CapabilityMover_MAXClimbRate() {return
SUP_CapabilityMover_MAXClimbRate;}
```

```
     // Set Methods
     void SUP_Set_CapabilityMover_Name(const char* Name)
{strcpy(SUP_CapabilityMover_Name, Name);}
     void SUP_Set_CapabilityMover_CommitAlt(const char* CommitAlt)
{strcpy(SUP_CapabilityMover_CommitAlt, CommitAlt);}
     void SUP_Set_CapabilityMover_FuelUsage(const char* FuelUsage)
{strcpy(SUP_CapabilityMover_FuelUsage, FuelUsage);}
     void SUP_Set_CapabilityMover_NavErrorData(const char* NavErrorData)
{strcpy(SUP_CapabilityMover_NavErrorData, NavErrorData);}
     void SUP_Set_CapabilityMover_MAXAcceleration(float MAXAcceleration)
{SUP_CapabilityMover_MAXAcceleration = MAXAcceleration;}
     void SUP_Set_CapabilityMover_MINAltitude(float MINAltitude)
{SUP_CapabilityMover_MINAltitude = MINAltitude;}
     void SUP_Set_CapabilityMover_MAXAltitude(float MAXAltitude)
{SUP_CapabilityMover_MAXAltitude = MAXAltitude;}
     void SUP_Set_CapabilityMover_MINTurnRadius(float MINTurnRadius)
{SUP_CapabilityMover_MINTurnRadius = MINTurnRadius;}
     void SUP_Set_CapabilityMover_MINSpeed(float MINSpeed) {SUP_CapabilityMover_MINSpeed
= MINSpeed;}
     void SUP_Set_CapabilityMover_MAXSpeed(float MAXSpeed) {SUP_CapabilityMover_MAXSpeed
= MAXSpeed;}
     void SUP_Set_CapabilityMover_MAXDiveRate(float MAXDiveRate)
{SUP_CapabilityMover_MAXDiveRate = MAXDiveRate;}
     void SUP_Set_CapabilityMover_MAXClimbRate(float MAXClimbRate)
{SUP_CapabilityMover_MAXClimbRate = MAXClimbRate;}

     // Constructor and Destructor Methods
     SUP_CapabilityMover(const char* Name, const char* CommitAlt, const char* FuelUsage,
const char* NavErrorData, float MAXAcceleration, float MINAltitude, float MAXAltitude,
float MINTurnRadius, float MINSpeed, float MAXSpeed, float MAXDiveRate, float
MAXClimbRate);
     ~SUP_CapabilityMover();

    static os_typespec* get_os_typespec();

private:
     char SUP_CapabilityMover_Name[31];
     char SUP_CapabilityMover_CommitAlt[31];
     char SUP_CapabilityMover_FuelUsage[31];
     char SUP_CapabilityMover_NavErrorData[51];
     float SUP_CapabilityMover_MAXAcceleration;
     float SUP_CapabilityMover_MINAltitude;
     float SUP_CapabilityMover_MAXAltitude;
     float SUP_CapabilityMover_MINTurnRadius;
     float SUP_CapabilityMover_MINSpeed;
     float SUP_CapabilityMover_MAXSpeed;
     float SUP_CapabilityMover_MAXDiveRate;
     float SUP_CapabilityMover_MAXClimbRate;
};
```

5.3.3. Integrating metadata

The exported schema must be compared to the global schema and all schema conflicts must be solved. This comparison is the first step of the methodology defined in Chapter 3.

Step 1: The classes that belong to the SUP_Mover hierarchy were semantically compared to the global schema. The SUP_Mover classes had no representation in the global schema but the SUP_MoverCapability has the same semantics as Airplanes and Helicopters. The schema conflicts detected were the following:

Conflict 1: Missing related classes for SUP_Mover, SUP_MoverAirplane and SUP_Mover Helicopter.

Conflict 2: The attribute SUP_Mover_Name of class SUP_Mover, defined as string, is related to the Airframe_ID of class Airframe in the global schema.

Conflict 3: SUP_MoverCapability has the following semantically related attributes:

- SUP_MoverCapability_MAXAccelelation: This attribute is related to Aircraft_MAXG and both have the same representation.
- SUP_MoverCapability_MINSpeed: This attribute is related to Aircraft_MINSpeed, but they have different representations. In the exported suppressor schema it is represented by float and in the global schema it is represented by integer.
- SUP_MoverCapability_MAXSpeed: This attribute is related to Aircraft_MAXSpeed, but they have different representations. In the exported suppressor schema it is represented by float and in the global schema it is represented by integer.
- SUP_MoverCapability_MAXClimbRate: This attribute is related to Aircraft_MAX ClimbAngle, but they have different representations. In the exported suppressor schema it is represented by float and in the global schema it is represented by integer.

Conflict 4: The following attributes belong to the class SUP_MoverCapability and don't have representation in the semantically related class in the global schema:

- SUP_MoverCapability_Name
- SUP_MoverCapability_CommitAlt
- SUP_MoverCapability_FuelUsage
- SUP_MoverCapability_NavErrorData
- SUP_MoverCapability_MINTurnRadius
- SUP_MoverCapability_MINAltitude
- SUP_MoverCapability_MAXAltitude
- SUP_MoverCapability_MAXDiveRate

Conflict 5: All methods of the missing classes are missing and all methods that are related to the missing attributes are also missing.

Conflict 6: All methods that are related to semantically similar attributes have different names.

Step 2: The conflicts between MoverCapability and Airplanes/Helicopters need to be solved. The resolution for the conflicts detected above are shown below:

Resolution for conflict 1: Create the classes Mover, MoverAirplane and MoverHelicopter related to SUP_Mover, SUP_MoverAirplane and SUP_MoverHelicopter, respectively.

Resolution for conflict 2: The attribute SUP_Mover_Name, constant of the class SUP_Mover, is related to Airframe_ID of class Airframe in the global schema. Therefore it will exist in the class Mover. Create new methods Set_Mover_Name and Get_Mover_Name for class Mover. These methods are semantically related to SUP_Set_Mover_Name and SUP_Get_Mover_Name respectively and will manipulate the Airframe_ID attribute value in the class Airframe.

Resolution for conflict 3: Insert the following attribute-aliases:

- SUP_MoverCapability_MAXAccelelation for attribute Aircraft_MAXG
- SUP_MoverCapability_MINSpeed for attribute Aircraft_MINSpeed
- SUP_MoverCapability_MAXSpeed for attribute Aircraft_MAXSpeed
- SUP_MoverCapability_MAXClimbRate for attribute Aircraft_MAXClimbAngle

The three latter attributes have different representations than the related attributes in the global schema. As the float representation is more generic than the integer representation, the attribute type integer of these three attributes in the global schema needs to be changed to float. As integer and float data types have automatic transformation in the Object-Store database, no new method needs to be implemented to perform this transformation.

Resolution for conflict 4: Create the following attributes in the class Aircraft, since this class is the parent class of both Airplane and Helicopter classes:

- Aircraft_Name to represent SUP_MoverCapability_Name
- Aircraft_ CommitAlt to represent SUP_MoverCapability_CommitAlt

- Aircraft_ FuelUsage to represent SUP_MoverCapability_FuelUsage
- Aircraft_ NavErrorData to represent SUP_MoverCapability_NavErrorData
- Aircraft_ MINTurnRadius to represent SUP_MoverCapability_MINTurnRadius
- Aircraft_ MINAltitude to represent SUP_MoverCapability_MINAltitude
- Aircraft_ MAXAltitude to represent SUP_MoverCapability_MAXAltitude
- Aircraft_ MAXDiveRate to represent SUP_MoverCapability_MAXDiveRate

Resolution for conflict 5: Create all missing methods for the newly created classes and attributes.

Resolution for conflict 6: Insert the method-aliases:

- SUP_MoverCapability_MAXAccelelation for attribute Aircraft_MAXG
- SUP_MoverCapability_MINSpeed for attribute Aircraft_MINSpeed
- SUP_MoverCapability_MAXSpeed for attribute Aircraft_MAXSpeed
- SUP_MoverCapability_MAXClimbRate for attribute Aircraft_MAXClimbAngle

Step 3: Apply the resolutions described in the previous step for metadata information modifying information in the Integration dictionary to reflect the integration. The new global object model is shown in Figure 21.

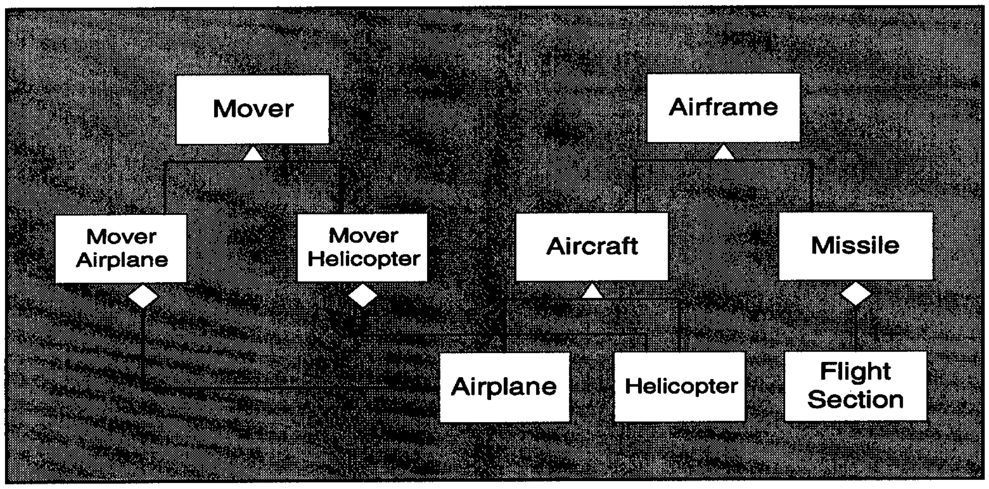


Figure 21: Global Object Model after integration

5.3.4. Integrating schema

Step 4: Change the global schema code in order to reflect the Suppressor exported schema integration.

The body codes of the transformed global classes' schemas (Aircraft, Airplane and Helicopter) are shown in Appendix C, Section 2. The header codes of the transformed global classes' schemas are shown in the following descriptions:

Aircraft

Aircraft header

```
class Aircraft : Airframe
{
public:
     // Get Method call parent
     const char* Get_Aircraft_ID() {return Airframe::Get_Airframe_ID();}

     // Get Methods
     const char* Get_Aircraft_CapabilityName() {return Aircraft_CapabilityName;}
     const char* Get_Aircraft_CommitAlt() {return Aircraft_CommitAlt;}
     const char* Get_Aircraft_FuelUsage() {return Aircraft_FuelUsage;}
     const char* Get_Aircraft_NavErrorData() {return Aircraft_NavErrorData;}
     int Get_Aircraft_NonAerodynamic() {return Aircraft_NonAerodynamic;}
     float Get_Aircraft_MAXSpeed() {return Aircraft_MAXSpeed;}
     float Get_Aircraft_MINSpeed() {return Aircraft_MINSpeed;}
     float Get_Aircraft_MAXG() {return Aircraft_MAXG;}
     int Get_Aircraft_EmptyWeight() {return Aircraft_EmptyWeight;}
     int Get_Aircraft_FuelWeight() {return Aircraft_FuelWeight;}
     int Get_Aircraft_RTBFuelBingoLimit() {return Aircraft_RTBFuelBingoLimit;}
     int Get_Aircraft_AirRefuelBingoLimit() {return Aircraft_AirRefuelBingoLimit;}
     int Get_Aircraft_MaxFuelReceivingRate() {return Aircraft_MaxFuelReceivingRate;}
     float Get_Aircraft_LookAheadInterval() {return Aircraft_LookAheadInterval;}
     float Get_Aircraft_MultiplicationFactor() {return Aircraft_MultiplicationFactor;}
     int Get_Aircraft_TerrainSamples() {return Aircraft_TerrainSamples;}
     int Get_Aircraft_FeedbackControlGain() {return Aircraft_FeedbackControlGain;}
     float Get_Aircraft_MAXClimbAngle() {return Aircraft_MAXClimbAngle;}
     float Get_Aircraft_MINTurnRadius() {return Aircraft_MINTurnRadius;}
     float Get_Aircraft_MINAltitude() {return Aircraft_MINAltitude;}
     float Get_Aircraft_MAXAltitude() {return Aircraft_MAXAltitude;}
     float Get_Aircraft_MAXDiveRate() {return Aircraft_MAXDiveRate;}

     // Set Methods from the parent
     void Set_Aircraft_ID(const char* ID) {Airframe::Set_Airframe_ID(ID);}

     // Set Methods
     void Set_Aircraft_CapabilityName(const char* CapabilityName)
{strcpy(Aircraft_CapabilityName, CapabilityName);}
     void Set_Aircraft_CommitAlt(const char* CommitAlt) {strcpy(Aircraft_CommitAlt,
CommitAlt);}
     void Set_Aircraft_FuelUsage(const char* FuelUsage) {strcpy(Aircraft_FuelUsage,
FuelUsage);}
     void Set_Aircraft_NavErrorData(const char* NavErrorData)
{strcpy(Aircraft_NavErrorData, NavErrorData);}
     void Set_Aircraft_NonAerodynamic(int NonAerodynamic) {Aircraft_NonAerodynamic =
NonAerodynamic;}
     void Set_Aircraft_MAXSpeed(float MAXSpeed) {Aircraft_MAXSpeed = MAXSpeed;}
     void Set_Aircraft_MINSpeed(float MINSpeed) {Aircraft_MINSpeed = MINSpeed;}
     void Set_Aircraft_MAXG(float MAXG) {Aircraft_MAXG = MAXG;}
     void Set_Aircraft_EmptyWeight(int EmptyWeight) {Aircraft_EmptyWeight = EmptyWeight;}
     void Set_Aircraft_FuelWeight(int FuelWeight) {Aircraft_FuelWeight = FuelWeight;}
     void Set_Aircraft_RTBFuelBingoLimit(int RTBFuelBingoLimit)
{Aircraft_RTBFuelBingoLimit = RTBFuelBingoLimit;}
     void Set_Aircraft_AirRefuelBingoLimit(int AirRefuelBingoLimit)
{Aircraft_AirRefuelBingoLimit = AirRefuelBingoLimit;}
     void Set_Aircraft_MaxFuelReceivingRate(int MaxFuelReceivingRate)
{Aircraft_MaxFuelReceivingRate = MaxFuelReceivingRate;}
     void Set_Aircraft_LookAheadInterval(float LookAheadInterval)
{Aircraft_LookAheadInterval = LookAheadInterval;}
     void Set_Aircraft_MultiplicationFactor(float MultiplicationFactor)
{Aircraft_MultiplicationFactor = MultiplicationFactor;}
     void Set_Aircraft_TerrainSamples(int TerrainSamples) {Aircraft_TerrainSamples =
TerrainSamples;}
     void Set_Aircraft_FeedbackControlGain(int FeedbackControlGain)
{Aircraft_FeedbackControlGain = FeedbackControlGain;}
     void Set_Aircraft_MAXClimbAngle(float MAXClimbAngle) {Aircraft_MAXClimbAngle =
MAXClimbAngle;}
     void Set_Aircraft_MINTurnRadius(float MINTurnRadius) {Aircraft_MINTurnRadius =
MINTurnRadius;}
     void Set_Aircraft_MINAltitude(float MINAltitude) {Aircraft_MINAltitude =
MINAltitude;}
     void Set_Aircraft_MAXAltitude(float MAXAltitude) {Aircraft_MAXAltitude =
MAXAltitude;}
     void Set_Aircraft_MAXDiveRate(float MAXDiveRate) {Aircraft_MAXDiveRate =
MAXDiveRate;}

     // Show method
     void Aircraft_Show(ostream& os);

     // Constructor and Destructor Methods
     // suppressor
     Aircraft(const char* ID, const char* CapabilityName, const char* CommitAlt, const
char* FuelUsage, const char* NavErrorData, float MAXSpeed, float MINSpeed, float MAXG,
float MAXClimbAngle, float MINTurnRadius, float MINAltitude, float MAXAltitude, float
MAXDiveRate);
```

```
     // eadsim
     Aircraft(const char* ID, int NonAerodynamic, int MAXSpeed, int MINSpeed, float MAXG,
int EmptyWeight, int FuelWeight, int RTBFuelBingoLimit, int AirRefuelBingoLimit, int
MaxFuelReceivingRate, float LookAheadInterval, float MultiplicationFactor, int
TerrainSamples, int FeedbackControlGain, int MAXClimbAngle);

     //general
     Aircraft(const char* ID, const char* CapabilityName, const char* CommitAlt, const
char* FuelUsage, const char* NavErrorData, int NonAerodynamic, float MAXSpeed, float
MINSpeed, float MAXG, int EmptyWeight, int FuelWeight, int RTBFuelBingoLimit, int
AirRefuelBingoLimit, int MaxFuelReceivingRate, float LookAheadInterval, float
MultiplicationFactor, int TerrainSamples, int FeedbackControlGain, float MAXClimbAngle,
float MINTurnRadius, float MINAltitude, float MAXAltitude, float MAXDiveRate);

     ~Aircraft();

protected:
     char Aircraft_CapabilityName[31];
     char Aircraft_CommitAlt[31];
     char Aircraft_FuelUsage[31];
     char Aircraft_NavErrorData[51];
     int Aircraft_NonAerodynamic;
     float Aircraft_MAXSpeed;
     float Aircraft_MINSpeed;
     float Aircraft_MAXG;
     int Aircraft_EmptyWeight;
     int Aircraft_FuelWeight;
     int Aircraft_RTBFuelBingoLimit;
     int Aircraft_AirRefuelBingoLimit;
     int Aircraft_MaxFuelReceivingRate;
     float Aircraft_LookAheadInterval;
     float Aircraft_MultiplicationFactor;
     int Aircraft_TerrainSamples;
     int Aircraft_FeedbackControlGain;
     float Aircraft_MAXClimbAngle;
     float Aircraft_MINTurnRadius;
     float Aircraft_MINAltitude;
     float Aircraft_MAXAltitude;
     float Aircraft_MAXDiveRate;
};
```

Airplane

Airplane header

```
class Airplane : Aircraft
{
public:
     // Get Methods call parent
     const char* Get_Airplane_ID() {return Aircraft::Get_Aircraft_ID();}
     const char* Get_Airplane_CapabilityName() {return
Aircraft::Get_Aircraft_CapabilityName();}
     const char* Get_Airplane_CommitAlt() {return Aircraft::Get_Aircraft_CommitAlt();}
     const char* Get_Airplane_FuelUsage() {return Aircraft::Get_Aircraft_FuelUsage();}
     const char* Get_Airplane_NavErrorData() {return
Aircraft::Get_Aircraft_NavErrorData();}
     int Get_Airplane_NonAerodynamic() {return Aircraft::Get_Aircraft_NonAerodynamic();}
     float Get_Airplane_MAXSpeed() {return Aircraft::Get_Aircraft_MAXSpeed();}
     float Get_Airplane_MINSpeed() {return Aircraft::Get_Aircraft_MINSpeed();}
     float Get_Airplane_MAXG() {return Aircraft::Get_Aircraft_MAXG();}
     int Get_Airplane_EmptyWeight() {return Aircraft::Get_Aircraft_EmptyWeight();}
     int Get_Airplane_FuelWeight() {return Aircraft::Get_Aircraft_FuelWeight();}
     int Get_Airplane_RTBFuelBingoLimit() {return
Aircraft::Get_Aircraft_RTBFuelBingoLimit();}
```

```
     int Get_Airplane_AirRefuelBingoLimit() {return
Aircraft::Get_Aircraft_AirRefuelBingoLimit();}
     int Get_Airplane_MaxFuelReceivingRate() {return
Aircraft::Get_Aircraft_MaxFuelReceivingRate();}
     float Get_Airplane_LookAheadInterval() {return
Aircraft::Get_Aircraft_LookAheadInterval();}
     float Get_Airplane_MultiplicationFactor() {return
Aircraft::Get_Aircraft_MultiplicationFactor();}
     int Get_Airplane_TerrainSamples() {return Aircraft::Get_Aircraft_TerrainSamples();}
     int Get_Airplane_FeedbackControlGain() {return
Aircraft::Get_Aircraft_FeedbackControlGain();}
     float Get_Airplane_MAXClimbAngle() {return Aircraft::Get_Aircraft_MAXClimbAngle();}
     float Get_Airplane_MINTurnRadius() {return Aircraft::Get_Aircraft_MINTurnRadius();}
     float Get_Airplane_MINAltitude() {return Aircraft::Get_Aircraft_MINAltitude();}
     float Get_Airplane_MAXAltitude() {return Aircraft::Get_Aircraft_MAXAltitude();}
     float Get_Airplane_MAXDiveRate() {return Aircraft::Get_Aircraft_MAXDiveRate();}

     // Get Methods
     int Get_Airplane_ABSpeed() {return Airplane_ABSpeed;}
     int Get_Airplane_MAXThrust() {return Airplane_MAXThrust;}
     float Get_Airplane_WingArea() {return Airplane_WingArea;}
     float Get_Airplane_WingSpan() {return Airplane_WingSpan;}
     int Get_Airplane_CruiseAltMSL() {return Airplane_CruiseAltMSL;}
     int Get_Airplane_CruiseSpeed() {return Airplane_CruiseSpeed;}
     int Get_Airplane_FuelFlow() {return Airplane_FuelFlow;}
     float Get_Airplane_TSFC() {return Airplane_TSFC;}
     int Get_Airplane_MAXSpeedCruiseAlt() {return Airplane_MAXSpeedCruiseAlt;}
     float Get_Airplane_CD0() {return Airplane_CD0;}

     // Set Methods from the parent
     void Set_Airplane_ID(const char* ID) {Aircraft::Set_Aircraft_ID(ID);}
     void Set_Airplane_CapabilityName(const char* CapabilityName)
{Aircraft::Set_Aircraft_CapabilityName(CapabilityName);}
     void Set_Airplane_CommitAlt(const char* CommitAlt)
{Aircraft::Set_Aircraft_CommitAlt(CommitAlt);}
     void Set_Airplane_FuelUsage(const char* FuelUsage)
{Aircraft::Set_Aircraft_FuelUsage(FuelUsage);}
     void Set_Airplane_NavErrorData(const char* NavErrorData)
{Aircraft::Set_Aircraft_NavErrorData(NavErrorData);}
     void Set_Airplane_NonAerodynamic(int NonAerodynamic)
{Aircraft::Set_Aircraft_NonAerodynamic(NonAerodynamic);}

     void Set_Airplane_MAXSpeed(float MAXSpeed)
{Aircraft::Set_Aircraft_MAXSpeed(MAXSpeed);}

     void Set_Airplane_MINSpeed(float MINSpeed)
{Aircraft::Set_Aircraft_MINSpeed(MINSpeed);}
     void Set_Airplane_MAXG(float MAXG) {Aircraft::Set_Aircraft_MAXG(MAXG);}
     void Set_Airplane_EmptyWeight(int EmptyWeight)
{Aircraft::Set_Aircraft_EmptyWeight(EmptyWeight);}
     void Set_Airplane_FuelWeight(int FuelWeight)
{Aircraft::Set_Aircraft_FuelWeight(FuelWeight);}
     void Set_Airplane_RTBFuelBingoLimit(int RTBFuelBingoLimit)
{Aircraft::Set_Aircraft_RTBFuelBingoLimit(RTBFuelBingoLimit);}
     void Set_Airplane_AirRefuelBingoLimit(int AirRefuelBingoLimit)
{Aircraft::Set_Aircraft_AirRefuelBingoLimit(AirRefuelBingoLimit);}
     void Set_Airplane_MaxFuelReceivingRate(int MaxFuelReceivingRate)
{Aircraft::Set_Aircraft_MaxFuelReceivingRate(MaxFuelReceivingRate);}
     void Set_Airplane_LookAheadInterval(float LookAheadInterval)
{Aircraft::Set_Aircraft_LookAheadInterval(LookAheadInterval);}
     void Set_Airplane_MultiplicationFactor(float MultiplicationFactor)
{Aircraft::Set_Aircraft_MultiplicationFactor(MultiplicationFactor);}
     void Set_Airplane_TerrainSamples(int TerrainSamples)
{Aircraft::Set_Aircraft_TerrainSamples(TerrainSamples);}
     void Set_Airplane_FeedbackControlGain(int FeedbackControlGain)
{Aircraft::Set_Aircraft_FeedbackControlGain(FeedbackControlGain);}
     void Set_Airplane_MAXClimbAngle(float MAXClimbAngle)
{Aircraft::Set_Aircraft_MAXClimbAngle(MAXClimbAngle);}
     void Set_Airplane_MINTurnRadius(float MINTurnRadius)
{Aircraft::Set_Aircraft_MINTurnRadius(MINTurnRadius);}
```

```
     void Set_Airplane_MINAltitude(float MINAltitude)
{Aircraft::Set_Aircraft_MINAltitude(MINAltitude);}
     void Set_Airplane_MAXAltitude(float MAXAltitude)
{Aircraft::Set_Aircraft_MAXAltitude(MAXAltitude);}

     void Set_Airplane_MAXDiveRate(float MAXDiveRate)
{Aircraft::Set_Aircraft_MAXDiveRate(MAXDiveRate);}

     // Set Methods
     void Set_Airplane_ABSpeed(int ABSpeed) {Airplane_ABSpeed = ABSpeed;}
     void Set_Airplane_MAXThrust(int MAXThrust) {Airplane_MAXThrust = MAXThrust;}
     void Set_Airplane_WingArea(float WingArea) {Airplane_WingArea = WingArea;}
     void Set_Airplane_WingSpan(float WingSpan) {Airplane_WingSpan = WingSpan;}
     void Set_Airplane_CruiseAltMSL(int CruiseAltMSL) {Airplane_CruiseAltMSL =
CruiseAltMSL;}
     void Set_Airplane_CruiseSpeed(int CruiseSpeed) {Airplane_CruiseSpeed = CruiseSpeed;}
     void Set_Airplane_FuelFlow(int FuelFlow) {Airplane_FuelFlow = FuelFlow;}
     void Set_Airplane_TSFC(float TSFC) {Airplane_TSFC = TSFC;}
     void Set_Airplane_MAXSpeedCruiseAlt(int MAXSpeedCruiseAlt)
{Airplane_MAXSpeedCruiseAlt = MAXSpeedCruiseAlt;}
     void Set_Airplane_CD0(float CD0) {Airplane_CD0 = CD0;}

     // Show method
     void Airplane_Show(ostream& os);

     // Constructor and Destructor Methods

     // general
     Airplane(const char* ID, const char* CapabilityName, const char* CommitAlt, const
char* FuelUsage, const char* NavErrorData, int NonAerodynamic, float MAXSpeed, float
MINSpeed, float MAXG, int EmptyWeight, int FuelWeight, int RTBFuelBingoLimit, int
AirRefuelBingoLimit, int MaxFuelReceivingRate, float LookAheadInterval, float
MultiplicationFactor, int TerrainSamples, int FeedbackControlGain, float MAXClimbAngle,
float MINTurnRadius, float MINAltitude, float MAXAltitude, float MAXDiveRate, int
ABSpeed, int MAXThrust, float WingArea, float WingSpan, int CruiseAltMSL, int
CruiseSpeed, int FuelFlow, float TSFC, int MAXSpeedCruiseAlt, float CD0);

     //suppressor
     Airplane(const char* ID, const char* CapabilityName, const char* CommitAlt, const
char* FuelUsage, const char* NavErrorData, float MAXSpeed, float MINSpeed, float MAXG,
float MAXClimbAngle, float MINTurnRadius, float MINAltitude, float MAXAltitude, float
MAXDiveRate);

     //eadsim
     Airplane(const char* ID, int NonAerodynamic, int MAXSpeed, int MINSpeed, float MAXG,
int EmptyWeight, int FuelWeight, int RTBFuelBingoLimit, int AirRefuelBingoLimit, int
MaxFuelReceivingRate, float LookAheadInterval, float MultiplicationFactor, int
TerrainSamples, int FeedbackControlGain, int MAXClimbAngle, int ABSpeed, int MAXThrust,

float WingArea, float WingSpan, int CruiseAltMSL, int CruiseSpeed, int FuelFlow, float
TSFC, int MAXSpeedCruiseAlt, float CD0);

     ~Airplane();

     static os_typespec* get_os_typespec();
private:
     int Airplane_ABSpeed;
     int Airplane_MAXThrust;
     float Airplane_WingArea;
     float Airplane_WingSpan;
     int Airplane_CruiseAltMSL;
     int Airplane_CruiseSpeed;
     int Airplane_FuelFlow;
     float Airplane_TSFC;
     int Airplane_MAXSpeedCruiseAlt;
     float Airplane_CD0;
};
```

Helicopter

Helicopter header

```
class Helicopter : Aircraft
{
public:
     // Get Methods call parent
     const char* Get_Helicopter_ID() {return Aircraft::Get_Aircraft_ID();}
     const char* Get_Helicopter_CapabilityName() {return
Aircraft::Get_Aircraft_CapabilityName();}
     const char* Get_Helicopter_CommitAlt() {return Aircraft::Get_Aircraft_CommitAlt();}
     const char* Get_Helicopter_FuelUsage() {return Aircraft::Get_Aircraft_FuelUsage();}
     const char* Get_Helicopter_NavErrorData() {return
Aircraft::Get_Aircraft_NavErrorData();}
     int Get_Helicopter_NonAerodynamic() {return
Aircraft::Get_Aircraft_NonAerodynamic();}
     float Get_Helicopter_MAXSpeed() {return Aircraft::Get_Aircraft_MAXSpeed();}
     float Get_Helicopter_MINSpeed() {return Aircraft::Get_Aircraft_MINSpeed();}
     float Get_Helicopter_MAXG() {return Aircraft::Get_Aircraft_MAXG();}
     int Get_Helicopter_EmptyWeight() {return Aircraft::Get_Aircraft_EmptyWeight();}
     int Get_Helicopter_FuelWeight() {return Aircraft::Get_Aircraft_FuelWeight();}
     int Get_Helicopter_RTBFuelBingoLimit() {return
Aircraft::Get_Aircraft_RTBFuelBingoLimit();}
     int Get_Helicopter_AirRefuelBingoLimit() {return
Aircraft::Get_Aircraft_AirRefuelBingoLimit();}
     int Get_Helicopter_MaxFuelReceivingRate() {return
Aircraft::Get_Aircraft_MaxFuelReceivingRate();}
     float Get_Helicopter_LookAheadInterval() {return
Aircraft::Get_Aircraft_LookAheadInterval();}
     float Get_Helicopter_MultiplicationFactor() {return
Aircraft::Get_Aircraft_MultiplicationFactor();}
     int Get_Helicopter_TerrainSamples() {return
Aircraft::Get_Aircraft_TerrainSamples();}
     int Get_Helicopter_FeedbackControlGain() {return
Aircraft::Get_Aircraft_FeedbackControlGain();}
     float Get_Helicopter_MAXClimbAngle() {return
Aircraft::Get_Aircraft_MAXClimbAngle();}
     float Get_Helicopter_MINTurnRadius() {return
Aircraft::Get_Aircraft_MINTurnRadius();}
     float Get_Helicopter_MINAltitude() {return Aircraft::Get_Aircraft_MINAltitude();}
     float Get_Helicopter_MAXAltitude() {return Aircraft::Get_Aircraft_MAXAltitude();}
     float Get_Helicopter_MAXDiveRate() {return Aircraft::Get_Aircraft_MAXDiveRate();}

     // Get Method
     float Get_Helicopter_Power() {return Helicopter_Power;}
     float Get_Helicopter_PSFC() {return Helicopter_PSFC;}
     float Get_Helicopter_DecelarationG() {return Helicopter_DecelarationG;}
     int Get_Helicopter_Blades() {return Helicopter_Blades;}
     float Get_Helicopter_BladeRadius() {return Helicopter_BladeRadius;}
     float Get_Helicopter_BladeChord() {return Helicopter_BladeChord;}
     float Get_Helicopter_BladeCl() {return Helicopter_BladeCl;}
     float Get_Helicopter_BladeCD0() {return Helicopter_BladeCD0;}
     float Get_Helicopter_TipVelocity() {return Helicopter_TipVelocity;}
     float Get_Helicopter_FuselageArea() {return Helicopter_FuselageArea;}
     float Get_Helicopter_FuselageCd0() {return Helicopter_FuselageCd0;}

     // Set Methods from the parent
     void Set_Helicopter_ID(const char* ID) {Aircraft::Set_Aircraft_ID(ID);}
     void Set_Helicopter_CapabilityName(const char* CapabilityName)
{Aircraft::Set_Aircraft_CapabilityName(CapabilityName);}
     void Set_Helicopter_CommitAlt(const char* CommitAlt)
{Aircraft::Set_Aircraft_CommitAlt(CommitAlt);}
     void Set_Helicopter_FuelUsage(const char* FuelUsage)
{Aircraft::Set_Aircraft_FuelUsage(FuelUsage);}
     void Set_Helicopter_NavErrorData(const char* NavErrorData)
{Aircraft::Set_Aircraft_NavErrorData(NavErrorData);}
     void Set_Helicopter_NonAerodynamic(int NonAerodynamic)
{Aircraft::Set_Aircraft_NonAerodynamic(NonAerodynamic);}
```

```
     void Set_Helicopter_MAXSpeed(float MAXSpeed)
{Aircraft::Set_Aircraft_MAXSpeed(MAXSpeed);}
     void Set_Helicopter_MINSpeed(float MINSpeed)
{Aircraft::Set_Aircraft_MINSpeed(MINSpeed);}
     void Set_Helicopter_MAXG(float MAXG) {Aircraft::Set_Aircraft_MAXG(MAXG);}
     void Set_Helicopter_EmptyWeight(int EmptyWeight)
Aircraft::Set_Aircraft_EmptyWeight(EmptyWeight);}
     void Set_Helicopter_FuelWeight(int FuelWeight)
{Aircraft::Set_Aircraft_FuelWeight(FuelWeight);}
     void Set_Helicopter_RTBFuelBingoLimit(int RTBFuelBingoLimit)
{Aircraft::Set_Aircraft_RTBFuelBingoLimit(RTBFuelBingoLimit);}
     void Set_Helicopter_AirRefuelBingoLimit(int AirRefuelBingoLimit)
{Aircraft::Set_Aircraft_AirRefuelBingoLimit(AirRefuelBingoLimit);}
     void Set_Helicopter_MaxFuelReceivingRate(int MaxFuelReceivingRate)
{Aircraft::Set_Aircraft_MaxFuelReceivingRate(MaxFuelReceivingRate);}
     void Set_Helicopter_LookAheadInterval(float LookAheadInterval)
{Aircraft::Set_Aircraft_LookAheadInterval(LookAheadInterval);}
     void Set_Helicopter_MultiplicationFactor(float MultiplicationFactor)
{Aircraft::Set_Aircraft_MultiplicationFactor(MultiplicationFactor);}
     void Set_Helicopter_TerrainSamples(int TerrainSamples)
{Aircraft::Set_Aircraft_TerrainSamples(TerrainSamples);}
     void Set_Helicopter_FeedbackControlGain(int FeedbackControlGain)
{Aircraft::Set_Aircraft_FeedbackControlGain(FeedbackControlGain);}
     void Set_Helicopter_MAXClimbAngle(float MAXClimbAngle)
{Aircraft::Set_Aircraft_MAXClimbAngle(MAXClimbAngle);}
     void Set_Helicopter_MINTurnRadius(float MINTurnRadius)
{Aircraft::Set_Aircraft_MINTurnRadius(MINTurnRadius);}
     void Set_Helicopter_MINAltitude(float MINAltitude)
{Aircraft::Set_Aircraft_MINAltitude(MINAltitude);}
     void Set_Helicopter_MAXAltitude(float MAXAltitude)
{Aircraft::Set_Aircraft_MAXAltitude(MAXAltitude);}
     void Set_Helicopter_MAXDiveRate(float MAXDiveRate)
{Aircraft::Set_Aircraft_MAXDiveRate(MAXDiveRate);}

     // Set Methods
     void Set_Helicopter_Power(float Power) {Helicopter_Power = Power;}
     void Set_Helicopter_PSFC(float PSFC) {Helicopter_PSFC = PSFC;}
     void Set_Helicopter_DecelarationG(float DecelarationG) {Helicopter_DecelarationG =
DecelarationG;}
     void Set_Helicopter_Blades(int Blades) {Helicopter_Blades = Blades;}
     void Set_Helicopter_BladeRadius(float BladeRadius) {Helicopter_BladeRadius =
BladeRadius;}
     void Set_Helicopter_BladeChord(float BladeChord) {Helicopter_BladeChord =
BladeChord;}
     void Set_Helicopter_BladeC1(float BladeC1) {Helicopter_BladeC1 = BladeC1;}
     void Set_Helicopter_BladeCD0(float BladeCD0) {Helicopter_BladeCD0 = BladeCD0;}
     void Set_Helicopter_TipVelocity(float TipVelocity) {Helicopter_TipVelocity =
TipVelocity;}
     void Set_Helicopter_FuselageArea(float FuselageArea) {Helicopter_FuselageArea =
FuselageArea;}
     void Set_Helicopter_FuselageCd0(float FuselageCd0) {Helicopter_FuselageCd0 =
FuselageCd0;}

     // Constructor and Destructor Methods

     // general
     Helicopter(const char* ID, const char* CapabilityName, const char* CommitAlt, const
char* FuelUsage, const char* NavErrorData, int NonAerodynamic, float MAXSpeed, float
MINSpeed, float MAXG, int EmptyWeight, int FuelWeight, int RTBFuelBingoLimit, int
AirRefuelBingoLimit, int MaxFuelReceivingRate, float LookAheadInterval, float
MultiplicationFactor, int TerrainSamples, int FeedbackControlGain, float MAXClimbAngle,
float MINTurnRadius, float MINAltitude, float MAXAltitude, float MAXDiveRate, float
Power, float PSFC, float DecelarationG, int Blades, float BladeRadius, float BladeChord,
float BladeC1, float BladeCD0, float TipVelocity, float FuselageArea, float FuselageCd0);

     // suppressor
     Helicopter(const char* ID, const char* CapabilityName, const char* CommitAlt, const
char* FuelUsage, const char* NavErrorData, float MAXSpeed, float MINSpeed, float MAXG,
float MAXClimbAngle, float MINTurnRadius, float MINAltitude, float MAXAltitude, float
MAXDiveRate);
```

```
     // eadsim
     Helicopter(const char* ID, int NonAerodynamic, int MAXSpeed, int MINSpeed, float
MAXG, int EmptyWeight, int FuelWeight, int RTBFuelBingoLimit, int AirRefuelBingoLimit,
int MaxFuelReceivingRate, float LookAheadInterval, float MultiplicationFactor,
int TerrainSamples, int FeedbackControlGain, int MAXClimbAngle, float Power, float PSFC,
float DecelarationG, int Blades, float BladeRadius, float BladeChord, float BladeC1,
float BladeCD0, float TipVelocity, float FuselageArea, float FuselageCd0);

     ~Helicopter();

     // Show method
     void Helicopter_Show(ostream& os);

     static os_typespec* get_os_typespec();

private:
     float Helicopter_Power;
     float Helicopter_PSFC;
     float Helicopter_DecelarationG;
     int Helicopter_Blades;
     float Helicopter_BladeRadius;
     float Helicopter_BladeChord;
     float Helicopter_BladeC1;
     float Helicopter_BladeCD0;
     float Helicopter_TipVelocity;
     float Helicopter_FuselageArea;
     float Helicopter_FuselageCd0;

};
```

The header codes of the new global schema classes are as follows:

<u>Mover</u>

Mover header

```
class Mover
{
public:
     // Get Methods

     int Get_Mover_SystemID() {return Mover_SystemID;}
     const char* Get_Mover_Type() {return Mover_Type;}

     //Set Methods
     void Set_Mover_SystemID(int SystemID) {Mover_SystemID = SystemID;}
     void Set_Mover_Type(const char* MType) {strcpy(Mover_Type, MType);}

     // Show method.

     void Mover_Show(ostream& os);

     // Constructor and Destructor Methods
     Mover(int SystemID);
     ~Mover();

protected:
     int Mover_SystemID;
     char Mover_Type[51];
};
```

MoverAirplane

MoverAirplane header

```
class MoverAirplane : Mover
{
public:
   // Get Methods from parent
   int Get_MoverAirplane_SystemID() {return Mover::Get_Mover_SystemID();}
   const char* Get_MoverAirplane_Type() {return Mover::Get_Mover_Type();}


   // Get Methods
   const char* Get_MoverAirplane_Name();
   Airplane* Get_MoverAirplane_CapabilityAirplane() {return CapabilityAirplane;}

   // Set_Methods from parent
   void Set_MoverAirplane_SystemID(int SystemID) {Mover::Set_Mover_SystemID(SystemID);}
   void Set_MoverAirplane_Type(const char* MType) {Mover::Set_Mover_Type(MType);}

   // Set Methods
   void Set_MoverAirplane_CapabilityAirplane(Airplane* CapAirplane);
   void Set_MoverAirplane_Name(const char* Name);

   // Show method
   void MoverAirplane_Show(ostream& os);

   // Constructor and Destructor Methods

   MoverAirplane(int SystemID, Airplane* CapAirplane);
   ~MoverAirplane();

   static os_typespec* get_os_typespec();

private:
   Airplane* CapabilityAirplane;
};
```

MoverHelicopter

MoverHelicopter header

```
class MoverHelicopter : Mover
{
public:

   // Get Methods from parent
   int Get_MoverHelicopter_SystemID() {return Mover::Get_Mover_SystemID();}
   const char* Get_MoverHelicopter_Type() {return Mover::Get_Mover_Type();}

  // Get Methods
   Helicopter* Get_MoverHelicopter_CapabilityHelicopter() {return CapabilityHelicopter;}
   const char* Get_MoverHelicopter_Name();


   // Set_Methods from parent
   void Set_MoverHelicopter_SystemID(int SystemID) {Mover::Set_Mover_SystemID(SystemID);}
   void Set_MoverHelicopter_Type(const char* MType) {Mover::Set_Mover_Type(MType);}

   // Set_Methods
   void Set_MoverHelicopter_CapabilityHelicopter(Helicopter* CapHelicopter);
   void Set_MoverHelicopter_Name(const char* Name);
   // Show method
   void MoverHelicopter_Show(ostream& os);

   // Constructor and Destructor Methods

   MoverHelicopter(int SystemID, Helicopter* CapHelicopter);
   ~MoverHelicopter();

   static os_typespec* get_os_typespec();

private:
   Helicopter* CapabilityHelicopter;
};
```

<u>**Step 4 (continuation):**</u> Apply the resolutions described in the previous step for the schema code. Some schema transformations of old global schema to new global schema (Chapter 3 Section 3.4. – *Schema evolution operation*) require the IA to write global schema evolution programs. These programs perform update of the schema structure and data migration from the old schema to the new schema (if necessary). In Object-Store, programs are necessary for schema changes (references) even if there isn't any change in data.

5.3.5. Integrating data

<u>**Step4 (continuation):**</u> All data stored in the exported database must be moved to the global database. If there are similar instances of related objects with value conflicts, the DBA and IA must decide what values will be constant in the global schema. A program for moving the data must be written by the IA.

5.3.6. User's View definition

One important characteristic of object-oriented languages is that we can overload constructors. The constructors are used to create a new instance of a class. For each system component of the global schema, there is an associated constructor for related classes. In my methodology, this feature plays an important role since it avoids the changes of previously generated User's Views.

The new view is generated based on the output generated by the User's View Constructor program listed in Appendix A, Section 2. The code of the User's View translation program for Suppressor is shown in Appendix D.

5.4. Data verification

Using show methods (which print all the attribute values of a class instance) defined for each class of the global schema, a data view program was written manually to show the data stored in the global database and to compare it with the input data. The program code for verification is in Appendix E.

5.5. Summary

In this Chapter I created a global database example in order to validate my methodology for integrating scenario databases of simulation systems. The base classes were created when I integrated the first system – EADSIM. Since the global schema (until that moment) had no classes, no integration conflicts arose. All the classes, attributes and methods were practically copied into the global schema. The real integration work was performed when the second simulation system was integrated into the non-empty global schema. Then, the four steps of my proposed methodology (Chapter 3) were successfully applied, validating the methodology. Also, the integration of a new system using this methodology has shown that the previously created User's View translation program didn't need to be changed since the method signatures used by these programs didn't change. This important feature of the methodology led me to conclude that the integration of new Scenario Databases will not change the previously created User's View translation programs.

6. Conclusion

6.1. Meeting Objectives

This work was motivated by the need to have a unified database to support different simulation system scenarios. The fundamental motivation was to facilitate the data manipulation and integration. In this thesis I suggest the approach of creating a common data repository where all the scenarios' data is to be stored. In order to implement this approach, an object-oriented database is used since it allows integration of different application environments. However, object-oriented databases still do not support the implementation of views, which are necessary when the integration of different application data is performed. User's Views are the visualization of part of the data stored in the common repository related to a specific simulation system scenario, which is a component of this global database. While the idea of a common database represented in an OO-Model is not new, my research extended the OO-Model with a layer containing the metadata of the global data stored in a common database and User's View information. This extension is the Integration Dictionary. This layer is then used to integrate a new User's View into the common database and to filter the data from this common database to create a specific view for each simulation system component.

The structure of this OO-Model extension is shown in Chapter 3, where the methodology for integrating databases is described. In this methodology, the integration is based on common concepts stored in the extended layer. The common concept approach groups classes (of different scenarios), which are semantically similar. The approach also explicitly maintains the interrelationship between classes of simulation system scenarios. This extension (Integration Dictionary) facilitates the necessary translations (mappings) between specific scenarios and the common database. Also, the methodology shows how to solve schema conflicts during the integration process.

I presented an implementation of this solution using Object-Store database in Chapter 4, where I describe:

- the structure of the Integration Dictionary as well as its manipulation system;
- an association of each schema integration conflict with a function of this system; and
- a system that generates all the parameters necessary to construct a specific user's view.

The automatic view generation of specific simulation system scenarios in the OO-Model (User's View) from the global database could not be implemented due to lack of time, but could be the subject of future research.

I presented a validation of my methodology in Chapter 5, where parts of two simulation system scenarios were integrated into the common database. First EADSIM was integrated, followed by Suppressor. The mapping programs were written for both scenarios, but the schema evolution programs were not written since they are particular for each object-oriented language.

6.2. Conclusion

It was not an objective of this work to create a complete and accurate global schema for simulation system scenarios. Once the implementation phase began, it became readily apparent that the amount of work required would exceed what could be accomplished in the thesis development time. Instead, a global scenario sample was created to validate the methodology based on two simulation systems' scenarios.

The integration of the first simulation system scenario (EADSIM) was easily made since the database was empty and no conflict resolution was applied. After the EADSIM integration the global database schema had the necessary schema base to validate the methodology. The complexity of integrating the second simulation system was reduced by my Integration Dictionary approach. It reduces considerably the schema evolution work necessary to integrate another schema into the basic schema. Only the changes in the hierarchy chain and data types that are not automatically handled by the database require specific programs to migrate the stored data. The main advantage of this approach is that updates of user's view generation programs of simulation systems that were previously integrated into the global database are rarely necessary. It happens because the view generation program uses method's signatures defined for each class to generate the specific view, and these existing methods signatures are not required to change when a new database is integrated into the global database. The need to change the User's View arises when a global class is deleted or previously defined method's signatures are changed.

The Integrator Administrator is the person who applies the methodology in order to integrate new system scenarios. This person needs to have knowledge of the object-oriented database definition and manipulation language, which is to be used to implement the global database, as well as knowledge of the scenario's schemas of the simulation system components and the global scenario schema. The IA has an important role in the integration process, since a successful result depends on his or her work. Although this work can be done following the directions given in this thesis, it would be improved if it were performed automatically. However, no computer system can figure out by itself the intrinsic concepts hidden in the classes of any simulation system scenario. An interactive system to help the IA is the best that can be accomplished.

6.3. Recommendations

The following recommendations avoid changes in the User's View generation program:

- During the global schema evolution, only delete classes if it cannot be avoided since, once deleted, the programs that generate User's Views that use the related deleted classes must be changed.
- If new methods need to be created, do not use the same method names already defined in the global schema. Instead, create new signatures. Also, do not delete a method unless it is not used by any User's View translation program. This solution avoids changes in these programs that use the changed or deleted method signature(s).

6.4. Future work

Future work falls into two categories described below.

- Improvement of the user interface.

The improvement can be achieved by using a graphic user interface to manipulate data stored in the Integration Dictionary, as well as data stored in the global database.

- Automation of the User's View Constructor program.

The actual stored parameters work well for the manual translation, but for the automatic translation a connection of the constructors' parameters to methods of the

Integration Dictionary is necessary. However, all the parameters necessary for the automatic User's View generation can be stored in the Integration Dictionary.

In Chapter 5, the respective constructor for each system component (EADSIM and Suppressor) was implemented to generate the proper views. It was manually written in the respective User's View generation program as shown in Appendix D.

The Conceptual-Methods can have an aggregate attribute used for the method type "Constructor", which will be the set of all constructors for the related class. The constructors' names are the same for all of them. The difference will be the parameters and the associated access method. Therefore, at least one more class must be implemented to represent the aggregated objects. Figure 22 shows the proposed aggregation in the Integration Dictionary schema.

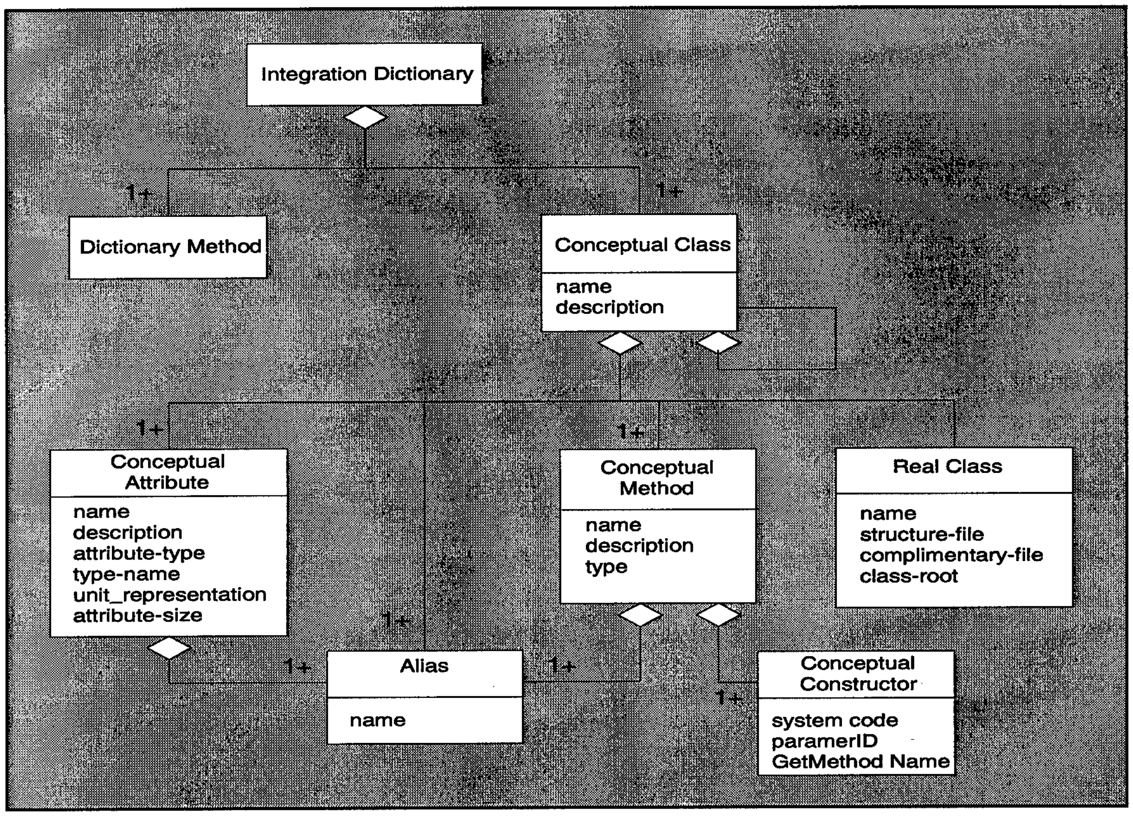


Figure 22: Integration Dictionary Model

Some necessary attributes (for this new class) are the following:

- System Code: this attribute stores the system code to which the construct belongs.
- For each parameter, the parameter identification and the related access method in the Integration Dictionary must be stored.

The algorithm must select all concrete classes from the Integration Dictionary that contain the required system code. From this set of classes the algorithm needs to identify the classes that have aggregations. The aggregate objects must be created first and then aggregated to the aggregation class. Also, this algorithm must have a solution for *joins* (join of two classes) to treat the case of an exported class to be broken into two or more global classes. Finally, the use of index structures should be considered in order to improve the queries' performance applied to the required selections of the Integration Dictionary data.

7. References

[1] Banarjee, J., Kim, W., Kim, N. K., and Korth, H.F., *"Semantics and Implementation of Schema Evolution in Object-Oriented Databases"*, in Proceedings of ACM –SIGMOD International Conference on Management of Data, San Francisco CA, May 1987.

[2] Bertino, E., *"A view mechanism for object-oriented databases"*, in 3rd International Conference on Extending Database Technology, p. 64-69, March 1992.

[3] Bertino, E., Negri, M., Pellagatti, G., Sbattella, L., "*Integration of Heterogeneous Database Applications through an Object-Oriented Interface*", Information Systems, Vol. 14, no. 5, p. 407-420, 1989.

[4] Bertino, E., Martino, L., "*Object-Oriented Database Systems – Concepts and Architectures*", Addison-Wesley, 1994.

[5] Blaha, M., Premerlani, W., *"A Catalog of Object Model Transformations"*, in 3rd Working Conference on Reverse Engineering, Monterey CA, November 1996.

[6] Cattel, R., and others, "*Object Database Standard: ODMG 2.0*" , Morgan Kaufmann, 1997.

[7] Defense Research Associates Inc., "*Collaborative Engineering Real Time Base Correlation tool*", Report Number AF97-136 prepared for DoD - Small Business Innovation Research (SBIR) Program, 1997.

[8] DoD, "*High-Level Architecture Object Model Template Specification*", Version 1.3, February 1998, accessible at http://hla.dmso.mil/hla/

[9] DoD, "*Modeling & Simulation Master Plan*", October 1995, accessible at http://www.dmso.mil/

[10] Hainaaut, J-L., Chandelon, M., Tonneau, C., Joris, M., "*Contribution to a Theory of Database Reverse Engineering*", Proc. of the IEEE Working Conference on Reverse Engineering, Baltimore, 1993.

[11] Hammer, J., McLeod D., Si, A., "*An Intelligent System for Identifying and Integrating Non-Local Objects in Federated Database Systems*", IEEE in Proceedings of 27th Annual Hawaii International Conference on Systems Science, 1994.

[12] Henrard, J., Englebert, V., Hick, J-M., Roland, D., and Hainaaut, J-L., "*Program understanding in database reverse engineering*", ACM journals, January 1998.

[13] Hull R., "*Managing Semantic Heterogeneity in Databases:A Theoretical Perspective*", in Proceedings of the 16th ACM SIGMOD – SIGART Symposium on Principles of Database Systems, p. 51-61, 1997.

[14] Institute for Simulation and Training, *"Standard for Distributed Interactive Simulation Protocols. Version 2.0 Fourth draft"*, Technical Report IST-CR-93-40, University of Florida, March 1994.

[15] Kim, W., "*Modern Database Systems : The Object Model, Interoperability, and Beyond*", Chapter 25, p.526-548, Addison Wesley, 1995.

[16] Kuno, H.A., Ra, Y.G., Rundensteiner, E.A., "*The Object-Slicing Technique: A flexible object representation and its evolution*", Technical Report CSCE-TR-241-95, University of Michigan, 1995.

[17] Linhares Lima, Pedro, "*Methodology for reengineering relational database to an object-oriented database*", Masters' Thesis AFIT/GCS/ENG/96J-01

[18] Object Design, Inc, "*Object Store Release 5.0 for all platforms*", Technical Manual Kit, March 1997.

[19] Object Management Group, "*CORBA/IIOP 2.2 Specification*", accessible at http://www.org.org/corba/corbiiop.htm

[20] Mullaney, J. P., "*A General Object Transformation System*", Masters' Thesis AFIT/ GCS/ENG/94D-18.

[21] Park, H.-G., "*The Development of a Scenario Translator for Distributed Simulations*", Masters' Thesis AFIT/GCS/ENG/96D-22.

[22] Pitoura, E., Burhes, O., Elmagarneid, A., "*Object Orientation Multidatabase*", ACM Computing Surveys, Vol. 17, no. 2, June 1995.

[23] Pope, A. R., *"The SIMNET Network and Protocols"*, Bolt and Newman Inc. Report Number 7627 prepared for Defense Advanced Research Projects Agency, June 1991.

[24] Quilici, Alex, "*Reverse Engineering of Legacy Systems: A Path Toward Success*", ICSE'95, Seattle - Washington, 1995.

[25] Ra, Y. G., Rundensteiner, E. A., "*A Transparent Object-Oriented Schema Change Approach using View Schema Evolution*", in IEEE International Conference on Data Engineering, p.165-172, March 1995.

[26] Rizza, R.J., "*An Object-Oriented Military Simulation Baseline for Parallel Simulation Research*", Masters' Thesis AFIT/GCS/ENG/90D-12.

[27] Rumbaugh, J. and others, "*Object-Oriented Modeling and Design*", Englewood Cliffs, New Jersey: Prentice Hall, 1991.

[28] Rundensteiner, E. A., "*Multiview: A Methodology for Supporting Multiple View Schemata in Object-Oriented Databases*", Technical Report 92-07, University of California, Irvine, 1992.

[29] Stratton, Phillip, "*A Metrics-based Analysis of Interface Usability Improvements by Applying Intelligent Agents*", Masters' Thesis AFIT/GCS/ENG/99M-18

[30] Teledyne Brown Engineering, "User's Manual for EADSIM – Version 5.01", 22 June 1996.

[31] Vidal, V., Winslett, M., "*Preserving Update Semantics in Schema Integration*", in Proceedings of the 3rd International Conference on Information and Knowledge Management, p. 263-271, 1994.

[32] Weber, Robert, "*A Methodology for extracting a Common Object Model for Simulation Systems*", Masters' Thesis AFIT/GCS/ENG/99M-20

Appendix A. View Constructor Program

Section 1: Output parameters for User's View Constructor program for EADSIM

Section 2: Output parameters for User's View Constructor program for Suppressor

Section 1. Output parameters for User's View Constructor program for EADSIM

```
conceptual classes used to generate view for EAD:

1)Airplane
  for exported class => EAD_Airplane


  methods of this class used to construct instance of class EAD_Airplane for system EAD:
Get_Airplane_WingArea
Get_Airplane_MAXG
Get_Airplane_MultiplicationFactor
Get_Airplane_MINSpeed
Get_Airplane_MaxFuelReceivingRate
Get_Airplane_LookAheadInterval
Get_Airplane_MAXSpeedCruiseAlt
Get_Airplane_MAXThrust
Get_Airplane_NonAerodynamic
Get_Airplane_AirRefuelBingoLimit
Get_Airplane_TSFC
Get_Airplane_TerrainSamples
Get_Airplane_EmptyWeight
Get_Airplane_WingSpan
Get_Airplane_CD0
Get_Airplane_MAXClimbAngle
Get_Airplane_MAXSpeed
Get_Airplane_ABSpeed
Get_Airplane_ID
Get_Airplane_FeedbackControlGain
Get_Airplane_CruiseAltMSL
Get_Airplane_FuelWeight
Get_Airplane_CruiseSpeed
Get_Airplane_RBTFuelBingoLimit
Get_Airplane_FuelFlow

  real class information for this class
=== Real Class Name: Airplane
    Real Class Root: Airplane_extent_root
    Structure File Name: Airframe.hh
    Complementary File Name: Airframe.cc

2)Helicopter
  for exported class => EAD_Helicopter


  methods of this class used to construct instance of class EAD_Helicopter for system
EAD:
Get_Helicopter_BladeCd0
Get_Helicopter_EmptyWeight
Get_Helicopter_FuelWeight
Get_Helicopter_RBTFuelBingoLimit
Get_Helicopter_AirRefuelBingoLimit
Get_Helicopter_TipVelocity
Get_Helicopter_MultiplicationFactor
Get_Helicopter_MaxFuelReceivingRate
Get_Helicopter_ID
Get_Helicopter_TerrainSamples
Get_Helicopter_FeedbackControlGain
Get_Helicopter_MINSpeed
Get_Helicopter_BladeRadius
Get_Helicopter_PSFC
Get_Helicopter_BladeChord
Get_Helicopter_DecelarationG
Get_Helicopter_BladeC1
Get_Helicopter_LookAheadInterval
Get_Helicopter_FuselageArea
Get_Helicopter_MAXClimbAngle
Get_Helicopter_NonAerodynamic
Get_Helicopter_MAXSpeed
Get_Helicopter_Blades
```

```
Get_Helicopter_Power
Get_Helicopter_FuselageCd0
Get_Helicopter_MAXG

  real class information for this class
=== Real Class Name: Helicopter
    Real Class Root: Helicopter_extent_root
    Structure File Name: Airframe.hh
    Complementary File Name: Airframe.cc

3)Missile
  for exported class => EAD_Missile

  ==> This class has the multivalued attribute Missile_FlightSections as aggregate
objects of set of type FlightSection
  ==> The multivalued object must be generated first in the exported schema, if it has
the same aggregation, and then aggregated to the instance of class EAD_Missile

  methods of this class used to construct instance of class EAD_Missile for system EAD:
Get_Missile_InitialVelocity
Get_Missile_LaunchRailLenght
Get_Missile_FlightSections
Get_Missile_ID
Get_Missile_LaunchElevationAngle
Get_Missile_MAXDivertVelocity
Get_Missile_MaxDivertLateralAcceleration

  real class information for this class
=== Real Class Name: Missile
    Real Class Root: Missile_extent_root
    Structure File Name: Airframe.hh
    Complementary File Name: Airframe.cc

4)FlightSection
  for exported class => EAD_FlightSection


  methods of this class used to construct instance of class EAD_FlightSection for system
EAD:
Get_FlightSection_SpecificImpulse
Get_FlightSection_ReferenceArea
Get_FlightSection_IntegrationTimeStep
Get_FlightSection_MAXG
Get_FlightSection_ResponseTime
Get_FlightSection_InitialMass
Get_FlightSection_IRSignature
Get_FlightSection_EndTime
Get_FlightSection_NozzleExitArea
Get_FlightSection_ProNavGuidanceGain
Get_FlightSection_MaxAlpha
Get_FlightSection_RCS

  real class information for this class
=== Real Class Name: FlightSection
    Real Class Root: FlightSection_extent_root
    Structure File Name: FlightSection.hh
    Complementary File Name: FlightSection.cc
```

Section 2. Output parameters for User's View Constructor program for Suppressor

conceptual classes used to generate view for SUP:

```
1)Airplane
  for exported class => SUP_CapabilityMover


  methods of this class used to construct instance of class SUP_CapabilityMover for
system SUP:
Get_Airplane_MAXG
Get_Airplane_NavErrorData
Get_Airplane_CommitAlt
Get_Airplane_MAXClimbAngle
Get_Airplane_MAXDiveRate
Get_Airplane_MINSpeed
Get_Airplane_MAXSpeed
Get_Airplane_CapabilityName
Get_Airplane_MINTurnRadius
Get_Airplane_FuelUsage
Get_Airplane_MINAltitude
Get_Airplane_MAXAltitude

  real class information for this class
=== Real Class Name: Airplane
    Real Class Root: Airplane_extent_root
    Structure File Name: Airframe.hh
    Complementary File Name: Airframe.cc

2)Helicopter
  for exported class => SUP_CapabilityMover


  methods of this class used to construct instance of class SUP_CapabilityMover for
system SUP:
Get_Helicopter_MAXDiveRate
Get_Helicopter_MINTurnRadius
Get_Helicopter_MAXClimbAngle
Get_Helicopter_FuelUsage
Get_Helicopter_MINAltitude
Get_Helicopter_MAXSpeed
Get_Helicopter_MAXAltitude
Get_Helicopter_MAXG
Get_Helicopter_CommitAlt
Get_Helicopter_CapabilityName
Set_Helicopter_MAXDiveRate
Get_Helicopter_MINSpeed
Get_Helicopter_NavErrorData

  real class information for this class
=== Real Class Name: Helicopter
    Real Class Root: Helicopter_extent_root
    Structure File Name: Airframe.hh
    Complementary File Name: Airframe.cc

3)MoverAirplane
  for exported class => SUP_MoverAirplane

  ==> This class has the attribute CapabilityAirplane as aggregate object of type
Airplane
  ==> The aggregate object must be generated first in the exported schema, if it has the
same aggregation, and then aggregated to the instance of class SUP_MoverAirplane

  methods of this class used to construct instance of class SUP_MoverAirplane for system
SUP:
Get_MoverAirplane_SystemID
Get_MoverAirplane_Name
Get_MoverAirplane_Type
Get_MoverAirplane_CapabilityAirplane
```

```
  real class information for this class
=== Real Class Name: MoverAirplane
    Real Class Root: MoverAirplane_extent_root
    Structure File Name: Mover.hh
    Complementary File Name: Mover.cc

4)MoverHelicopter
  for exported class => SUP_MoverHelicopter

  ==> This class has the attribute CapabilityHelicopter as aggregate object of type
Helicopter
  ==> The aggregate object must be generated first in the exported schema, if it has the
same aggregation, and then aggregated to the instance of class SUP_MoverHelicopter

  methods of this class used to construct instance of class SUP_MoverHelicopter for
system SUP:
Get_MoverHelicopter_SystemID
Get_MoverHelicopter_Name
Get_MoverHelicopter_Type
Get_MoverHelicopter_CapabilityHelicopter

  real class information for this class
=== Real Class Name: MoverHelicopter
    Real Class Root: MoverHelicopter_extent_root
    Structure File Name: Mover.hh
    Complementary File Name: Mover.cc
```

Appendix B. Body code of the exported classes' schemas

Section 1: Body code of the EADSIM exported classes' schema.

Section 2: Body code of the Suppressor exported classes' schema.

Section 1. Body code for EADSIM exported classes' schema

```
/*----------------------------------------------------------
--                  EAD_airframe.cc
--
--    Airframe Implementation for EADSIM exported schema
--
--                 Capt Emilia Colonese
----------------------------------------------------------*/

#include <string.h>
#include <ostore/ostore.hh>
#include <ostore/coll.hh>
#include "EAD_Airframe.hh"
#include "EAD_FlightSection.hh"

EAD_Airframe::EAD_Airframe(const char* Airframe_ID)
{
   strcpy(EAD_Airframe_ID, Airframe_ID);
}

EAD_Airframe::~EAD_Airframe()
{
}

EAD_Aircraft::EAD_Aircraft(const char* Airframe_ID, int NonAerodynamic, int MAXSpeed, int
MINSpeed, float MAXG, int EmptyWeight, int FuelWeight, int RTBFuelBingoLimit, int
AirRefuelBingoLimit, int MaxFuelReceivingRate, float LookAheadInterval, float
MultiplicationFactor, int TerrainSamples, int FeedbackControlGain, int MAXClimbAngle)
:EAD_Airframe(Airframe_ID)
{
   EAD_Aircraft_NonAerodynamic = NonAerodynamic;
   EAD_Aircraft_MAXSpeed = MAXSpeed;
   EAD_Aircraft_MINSpeed = MINSpeed;
   EAD_Aircraft_MAXG = MAXG;
   EAD_Aircraft_EmptyWeight = EmptyWeight;
   EAD_Aircraft_FuelWeight = FuelWeight;
   EAD_Aircraft_RTBFuelBingoLimit = RTBFuelBingoLimit;
   EAD_Aircraft_AirRefuelBingoLimit = AirRefuelBingoLimit;
   EAD_Aircraft_MaxFuelReceivingRate = MaxFuelReceivingRate;
   EAD_Aircraft_LookAheadInterval = LookAheadInterval;
   EAD_Aircraft_MultiplicationFactor = MultiplicationFactor;
   EAD_Aircraft_TerrainSamples = TerrainSamples;
   EAD_Aircraft_FeedbackControlGain = FeedbackControlGain;
   EAD_Aircraft_MAXClimbAngle = MAXClimbAngle;

}

EAD_Aircraft::~EAD_Aircraft()
{
}

EAD_Airplane::EAD_Airplane(const char* Airframe_ID, int NonAerodynamic, int MAXSpeed, int
MINSpeed, float MAXG, int EmptyWeight, int FuelWeight, int RTBFuelBingoLimit, int
AirRefuelBingoLimit, int MaxFuelReceivingRate, float LookAheadInterval, float
MultiplicationFactor, int TerrainSamples, int FeedbackControlGain, int MAXClimbAngle, int
ABSpeed, int MAXThrust, float WingArea, float WingSpan, int CruiseAltMSL, int
CruiseSpeed, int FuelFlow, float TSFC, int MAXSpeedCruiseAlt, float CD0)
:EAD_Aircraft(Airframe_ID, NonAerodynamic, MAXSpeed, MINSpeed, MAXG, EmptyWeight,
FuelWeight, RTBFuelBingoLimit, AirRefuelBingoLimit, MaxFuelReceivingRate,
LookAheadInterval, MultiplicationFactor, TerrainSamples, FeedbackControlGain,
MAXClimbAngle)
{
   EAD_Airplane_ABSpeed = ABSpeed;
   EAD_Airplane_MAXThrust = MAXThrust;
   EAD_Airplane_WingArea = WingArea;
   EAD_Airplane_WingSpan = WingSpan;
   EAD_Airplane_CruiseAltMSL = CruiseAltMSL;
   EAD_Airplane_CruiseSpeed = CruiseSpeed;
   EAD_Airplane_TSFC = TSFC;
   EAD_Airplane_MAXSpeedCruiseAlt = MAXSpeedCruiseAlt;
   EAD_Airplane_CD0 = CD0;
   EAD_Airplane_extent->insert(this);
}

EAD_Airplane::~EAD_Airplane()
{
  EAD_Airplane_extent->remove(this);
}

EAD_Helicopter::EAD_Helicopter(const char* Airframe_ID, int NonAerodynamic, int MAXSpeed,
int MINSpeed, float MAXG, int EmptyWeight, int FuelWeight, int RTBFuelBingoLimit, int
AirRefuelBingoLimit, int MaxFuelReceivingRate, float LookAheadInterval, float
MultiplicationFactor, int TerrainSamples, int FeedbackControlGain, int MAXClimbAngle,
float Power, float PSFC, float DecelarationG, int Blades, float BladeRadius, float
BladeChord, float BladeCl, float BladeCD0, float TipVelocity, float FuselageArea, float
FuselageCd0)
:EAD_Aircraft(Airframe_ID, NonAerodynamic, MAXSpeed, MINSpeed, MAXG, EmptyWeight,
FuelWeight, RTBFuelBingoLimit, AirRefuelBingoLimit, MaxFuelReceivingRate,
LookAheadInterval, MultiplicationFactor, TerrainSamples, FeedbackControlGain,
MAXClimbAngle)
{
   EAD_Helicopter_Power = Power;
   EAD_Helicopter_PSFC = PSFC;
   EAD_Helicopter_DecelarationG = DecelarationG;
   EAD_Helicopter_Blades = Blades;
   EAD_Helicopter_BladeRadius = BladeRadius;
   EAD_Helicopter_BladeChord = BladeChord;
   EAD_Helicopter_BladeCl = BladeCl;
   EAD_Helicopter_BladeCD0 = BladeCD0;
   EAD_Helicopter_TipVelocity = TipVelocity;
   EAD_Helicopter_FuselageArea = FuselageArea;
   EAD_Helicopter_FuselageCd0 = FuselageCd0;
   EAD_Helicopter_extent->insert(this);
}

EAD_Helicopter::~EAD_Helicopter()
{
  EAD_Helicopter_extent->remove(this);
}

void EAD_Missile::EAD_Set_Missile_FlightSection(EAD_FlightSection* FlighSection)
{
  EAD_Missile_FlightSections.insert(FlighSection);
}

EAD_Missile::EAD_Missile(const char* Airframe_ID, float InitialVelocity, float
LaunchRailLenght, float LaunchElevationAngle, float MAXDivertVelocity, float
MaxDivertLateralAcceleration)
:EAD_Airframe(Airframe_ID)
{
   EAD_Missile_InitialVelocity = InitialVelocity;
   EAD_Missile_LaunchRailLenght = LaunchRailLenght;
   EAD_Missile_LaunchElevationAngle = LaunchElevationAngle;
   EAD_Missile_MAXDivertVelocity = MAXDivertVelocity;
   EAD_Missile_MaxDivertLateralAcceleration = MaxDivertLateralAcceleration;
   EAD_Missile_extent->insert(this);
}

EAD_Missile::~EAD_Missile()
{
  EAD_Missile_extent->remove(this);
}
```

```
/*------------------------------------------------------------
--                EAD_FlightSection.cc
--
--  AD_FlightSection Implementation for EADSIM Exporte Schema
--
--             Capt Emilia Colonese
-----------------------------------------------------------*/
#include <string.h>
#include <ostore/ostore.hh>
#include <ostore/coll.hh>
#include "EAD_FlightSection.hh"

EAD_FlightSection::EAD_FlightSection(float EndTime, float MAXG, float SpecificImpulse,
float NozzleExitArea, float ReferenceArea, float InitialMass, float MaxAlpha, float
ResponseTime, float ProNavGuidanceGain, float IntegrationTimeStep, float IRSignature,
float RCS)
{

     EAD_FlightSection_EndTime = EndTime;
     EAD_FlightSection_MAXG = MAXG;
     EAD_FlightSection_SpecificImpulse = SpecificImpulse;
     EAD_FlightSection_NozzleExitArea = NozzleExitArea;
     EAD_FlightSection_ReferenceArea = ReferenceArea;
     EAD_FlightSection_InitialMass = InitialMass;
     EAD_FlightSection_MaxAlpha = MaxAlpha;
     EAD_FlightSection_ResponseTime = ResponseTime;
     EAD_FlightSection_ProNavGuidanceGain = ProNavGuidanceGain;
     EAD_FlightSection_IntegrationTimeStep = IntegrationTimeStep;
     EAD_FlightSection_IRSignature = IRSignature;
     EAD_FlightSection_RCS = RCS;
     EAD_FlightSection_extent->insert(this);
}

EAD_FlightSection::~EAD_FlightSection()
{
     EAD_FlightSection_extent->remove(this);
}
```

Section 2. Body code for Suppressor exported classes' schema

```
/*-----------------------------------------------------------
--                   SUP_Mover.cc
--
--   SUP_Mover Implementation for Suppressor exported schema
--
--                Capt Emilia Colonese
-----------------------------------------------------------*/

#include <string.h>
#include <stdlib.h>
#include <fstream.h>
#include <ostore/ostore.hh>
#include <ostore/coll.hh>
#include "SUP_Mover.hh"
#include "SUP_CapabilityMover.hh"

SUP_Mover::SUP_Mover(int SystemID, const char* Name)
{
  SUP_Mover_SystemID = SystemID;
  strcpy(SUP_Mover_Name, Name);
}

SUP_Mover::~SUP_Mover()
{
}

SUP_MoverAirplane::SUP_MoverAirplane(int SystemID, const char* MoverName,
SUP_CapabilityMover* CapMov)
:SUP_Mover(SystemID, MoverName)
{
   SUP_Set_Mover_Type("Airplane");
   SUP_MoverAirplane_CapabilityAirplane = CapMov;
   SUP_MoverAirplane_extent->insert(this);

}

SUP_MoverAirplane::~SUP_MoverAirplane()
{
   SUP_MoverAirplane_extent->remove(this);
}

SUP_MoverHelicopter::SUP_MoverHelicopter(int SystemID, const char* MoverName,
SUP_CapabilityMover* CapMov)
:SUP_Mover(SystemID, MoverName)
{
   SUP_Set_Mover_Type("Helicopter");
   SUP_MoverHelicopter_CapabilityHelicopter = CapMov;
   SUP_MoverHelicopter_extent->insert(this);
}

SUP_MoverHelicopter::~SUP_MoverHelicopter()
{
   SUP_MoverHelicopter_extent->remove(this);
}
```

```
/*----------------------------------------------------------------
--                    SUP_CapabilityMover.cc
--
-- SUP_CapabilityMover implementation for Suppressor exported schema
--
--                 Capt Emilia Colonese
--------------------------------------------------------------*/
#include <string.h>
#include <stdlib.h>
#include <fstream.h>
#include <ostore/ostore.hh>
#include <ostore/coll.hh>
#include "SUP_CapabilityMover.hh"

SUP_CapabilityMover::SUP_CapabilityMover(const char* Name, const char* CommitAlt, const
char* FuelUsage, const char* NavErrorData, float MAXAcceleration, float MINAltitude,
float MAXAltitude, float MINTurnRadius, float MINSpeed, float MAXSpeed, float
MAXDiveRate, float MAXClimbRate)
{
     strcpy(SUP_CapabilityMover_Name, Name);
     strcpy(SUP_CapabilityMover_CommitAlt, CommitAlt);
     strcpy(SUP_CapabilityMover_FuelUsage, FuelUsage);
     strcpy(SUP_CapabilityMover_NavErrorData, NavErrorData);
     SUP_CapabilityMover_MAXAcceleration = MAXAcceleration;
     SUP_CapabilityMover_MINAltitude = MINAltitude;
     SUP_CapabilityMover_MAXAltitude = MAXAltitude;
     SUP_CapabilityMover_MINTurnRadius = MINTurnRadius;
     SUP_CapabilityMover_MINSpeed = MINSpeed;
     SUP_CapabilityMover_MAXSpeed = MAXSpeed;
     SUP_CapabilityMover_MAXDiveRate = MAXDiveRate;
     SUP_CapabilityMover_MAXClimbRate = MAXClimbRate;
     SUP_CapabilityMover_extent->insert(this);
}

SUP_CapabilityMover::~SUP_CapabilityMover()
{
     SUP_CapabilityMover_extent->remove(this);
}
```

Appendix C. Body code of global classes' schemas

Section 1: Global classes after EADSIM integration.

Body code of the global classes' schemas after integrating EADSIM exported classes' schemas into the (empty) global schema.

Section 2: Global classes after Suppressor integration.

Body code of the global classes' schemas. After integrating Suppressor exported classes' schemas into the (non-empty) global schema.

Section 1. Global Classes after EADSIM integration

```
/*-----------------------------------------------------------
--                  Airframe.cc
--
--    Airframe Implementation for Global schema
--
--                Capt Emilia Colonese
-----------------------------------------------------------*/

#include <string.h>
#include <stdlib.h>
#include <fstream.h>
#include <ostore/ostore.hh>
#include <ostore/coll.hh>
#include "Airframe.hh"
#include "FlightSection.hh"

Airframe::Airframe(const char* ID)
{
   strcpy(Airframe_ID, ID);

  /*  Airframe_extent->insert(this);  This insertion cannot be implemented in the parents
of a hierarchy. This implementation must be made in the main program for insertion.  */
}

Airframe::~Airframe()
{
  Airframe_extent->remove(this);
}

void Airframe::Airframe_Show(ostream& os)
{
    os << "=== Airframe ID: " << Airframe_ID << endl;
}

Aircraft::Aircraft(char* ID, int NonAerodynamic, int MAXSpeed, int MINSpeed, float MAXG,
int EmptyWeight, int FuelWeight, int RTBFuelBingoLimit, int AirRefuelBingoLimit, int
MaxFuelReceivingRate, float LookAheadInterval, float MultiplicationFactor, int
TerrainSamples, int FeedbackControlGain, int MAXClimbAngle)
:Airframe(ID)
{
   Aircraft_NonAerodynamic = NonAerodynamic;
   Aircraft_MAXSpeed = MAXSpeed;
   Aircraft_MINSpeed = MINSpeed;
   Aircraft_MAXG = MAXG;
   Aircraft_EmptyWeight = EmptyWeight;
   Aircraft_FuelWeight = FuelWeight;
   Aircraft_RTBFuelBingoLimit = RTBFuelBingoLimit;
   Aircraft_AirRefuelBingoLimit = AirRefuelBingoLimit;
   Aircraft_MaxFuelReceivingRate = MaxFuelReceivingRate;
   Aircraft_LookAheadInterval = LookAheadInterval;
   Aircraft_MultiplicationFactor = MultiplicationFactor;
   Aircraft_TerrainSamples = TerrainSamples;
   Aircraft_FeedbackControlGain = FeedbackControlGain;
   Aircraft_MAXClimbAngle = MAXClimbAngle;
   /* Aircraft_extent->insert(this); This insertion cannot be implemented in the parents
of a hierarchy. This implementation must be made in the program for insertion.  */
}

Aircraft::~Aircraft()
{
   Aircraft_extent->remove(this);
}

void Aircraft::Aircraft_Show(ostream& os)
{
    Airframe::Airframe_Show(os);
        os << "===  NonAerodynamic: " << Aircraft_NonAerodynamic << endl;
    os << "===  MAXSpeed: " << Aircraft_MAXSpeed << endl;
    os << "===  MINSpeed: " << Aircraft_MINSpeed << endl;
    os << "===  MAXG: " << Aircraft_MAXG << endl;
    os << "===  EmptyWeight: " << Aircraft_EmptyWeight << endl;
    os << "===  FuelWeight: " << Aircraft_FuelWeight << endl;
    os << "===  RTBFuelBingoLimit: " << Aircraft_RTBFuelBingoLimit << endl;
    os << "===  AirRefuelBingoLimit: " << Aircraft_AirRefuelBingoLimit << endl;
    os << "===  MaxFuelReceivingRate: " << Aircraft_MaxFuelReceivingRate << endl;
    os << "===  LookAheadInterval: " << Aircraft_LookAheadInterval << endl;
    os << "===  MultiplicationFactor: " << Aircraft_MultiplicationFactor << endl;
    os << "===  TerrainSamples: " << Aircraft_TerrainSamples << endl;
    os << "===  FeedbackControlGain: " << Aircraft_FeedbackControlGain << endl;
    os << "===  MAXClimbAngle: " << Aircraft_MAXClimbAngle << endl;
}

Airplane::Airplane(char* ID, int NonAerodynamic, int MAXSpeed, int MINSpeed, float MAXG,
int EmptyWeight, int FuelWeight, int RTBFuelBingoLimit, int AirRefuelBingoLimit, int
MaxFuelReceivingRate, float LookAheadInterval, float MultiplicationFactor, int
TerrainSamples, int FeedbackControlGain, int MAXClimbAngle, int ABSpeed, int MAXThrust,
float WingArea, float WingSpan, int CruiseAltMSL, int CruiseSpeed, int FuelFlow, float
TSFC, int MAXSpeedCruiseAlt, float CD0)
:Aircraft(ID, NonAerodynamic, MAXSpeed, MINSpeed, MAXG, EmptyWeight, FuelWeight,
RTBFuelBingoLimit, AirRefuelBingoLimit, MaxFuelReceivingRate, LookAheadInterval,
MultiplicationFactor, TerrainSamples, FeedbackControlGain, MAXClimbAngle)
{
   Airplane_ABSpeed = ABSpeed;
   Airplane_MAXThrust = MAXThrust;
   Airplane_WingArea = WingArea;
   Airplane_WingSpan = WingSpan;
   Airplane_CruiseAltMSL = CruiseAltMSL;
   Airplane_CruiseSpeed = CruiseSpeed;
   Airplane_TSFC = TSFC;
   Airplane_MAXSpeedCruiseAlt = MAXSpeedCruiseAlt;
   Airplane_CD0 = CD0;
   Airplane_extent->insert(this);
}

Airplane::~Airplane()
{
   Airplane_extent->remove(this);
}

void Airplane::Airplane_Show(ostream& os)
{
    Aircraft::Aircraft_Show(os);
    os << "===  ABSpeed: " << Airplane_ABSpeed << endl;
    os << "===  MAXThrust: " << Airplane_MAXThrust << endl;
    os << "===  WingArea: " << Airplane_WingArea << endl;
    os << "===  WingSpan: " << Airplane_WingSpan << endl;
    os << "===  CruiseAltMSL: " << Airplane_CruiseAltMSL << endl;
    os << "===  CruiseSpeed: " << Airplane_CruiseSpeed << endl;
    os << "===  TSFC: " << Airplane_TSFC << endl;
    os << "===  MAXSpeedCruiseAlt: " << Airplane_MAXSpeedCruiseAlt << endl;
    os << "===  CD0: " << Airplane_CD0 << endl;
}

Helicopter::Helicopter(char* ID, int NonAerodynamic, int MAXSpeed, int MINSpeed, float
MAXG, int EmptyWeight, int FuelWeight, int RTBFuelBingoLimit, int AirRefuelBingoLimit,
int MaxFuelReceivingRate, float LookAheadInterval, float MultiplicationFactor, int
TerrainSamples, int FeedbackControlGain, int MAXClimbAngle, float Power, float PSFC,
float DecelarationG, int Blades, float BladeRadius, float BladeChord, float BladeCl,
float BladeCD0, float TipVelocity, float FuselageArea, float FuselageCd0)
:Aircraft(ID, NonAerodynamic, MAXSpeed, MINSpeed, MAXG, EmptyWeight, FuelWeight,
RTBFuelBingoLimit, AirRefuelBingoLimit, MaxFuelReceivingRate, LookAheadInterval,
MultiplicationFactor, TerrainSamples, FeedbackControlGain, MAXClimbAngle)
{
   Helicopter_Power = Power;
   Helicopter_PSFC = PSFC;
   Helicopter_DecelarationG = DecelarationG;
   Helicopter_Blades = Blades;
   Helicopter_BladeRadius = BladeRadius;
   Helicopter_BladeChord = BladeChord;
   Helicopter_BladeC1 = BladeC1;
   Helicopter_BladeCD0 = BladeCD0;
   Helicopter_TipVelocity = TipVelocity;
   Helicopter_FuselageArea = FuselageArea;
   Helicopter_FuselageCd0 = FuselageCd0;
   Helicopter_extent->insert(this);
}

Helicopter::~Helicopter()
{
   Helicopter_extent->remove(this);
}

void Helicopter::Helicopter_Show(ostream& os)
{
    Aircraft::Aircraft_Show(os);
        os << "===  Power: " << Helicopter_Power << endl;
    os << "===  PSFC: " << Helicopter_PSFC << endl;
    os << "===  DecelarationG: " << Helicopter_DecelarationG << endl;
    os << "===  Blades: " << Helicopter_Blades << endl;
    os << "===  BladeRadius: " << Helicopter_BladeRadius << endl;
    os << "===  BladeChord: " << Helicopter_BladeChord << endl;
    os << "===  BladeC1: " << Helicopter_BladeC1 << endl;
    os << "===  BladeCD0: " << Helicopter_BladeCD0 << endl;
    os << "===  TipVelocity: " << Helicopter_TipVelocity << endl;
    os << "===  FuselageArea: " << Helicopter_FuselageArea << endl;
    os << "===  TipVelocity: " << Helicopter_TipVelocity << endl;
    os << "===  FuselageCd0: " << Helicopter_FuselageCd0 << endl;
}

void Missile::Set_Missile_FlightSection(FlightSection* FlighSection)
{
  Missile_FlightSections.insert(FlighSection);
}

Missile::Missile(char* ID, float InitialVelocity, float LaunchRailLenght, float
LaunchElevationAngle, float MAXDivertVelocity, float MaxDivertLateralAcceleration)
:Airframe(ID)
{
   Missile_InitialVelocity = InitialVelocity;
   Missile_LaunchRailLenght = LaunchRailLenght;
   Missile_LaunchElevationAngle = LaunchElevationAngle;
   Missile_MAXDivertVelocity = MAXDivertVelocity;
   Missile_MaxDivertLateralAcceleration = MaxDivertLateralAcceleration;
   Missile_extent->insert(this);
}

Missile::~Missile()
{
   Missile_extent->remove(this);
}

// Show method
void Missile::Missile_Show(ostream& os)
{
    Airframe::Airframe_Show(os);
    os << "===  InitialVelocity: " << Missile_InitialVelocity << endl;
    os << "===  LaunchRailLenght: " << Missile_LaunchRailLenght << endl;
    os << "===  LaunchElevationAngle: " << Missile_LaunchElevationAngle << endl;
    os << "===  MAXDivertVelocity: " << Missile_MAXDivertVelocity << endl;
    os << "===  MaxDivertLateralAcceleration: " << Missile_MaxDivertLateralAcceleration
<< endl;
}
```

```
/*-------------------------------------------------------------
--                 FlightSection.cc
--
--    FlightSection Implementation for Global Schema
--
--                 Capt Emilia Colonese
-------------------------------------------------------------*/

#include <string.h>
#include <stdlib.h>
#include <fstream.h>
#include <ostore/ostore.hh>
#include <ostore/coll.hh>
#include "FlightSection.hh"

FlightSection::FlightSection(float EndTime, float MAXG, float SpecificImpulse, float
NozzleExitArea, float ReferenceArea, float InitialMass, float MaxAlpha, float
ResponseTime, float ProNavGuidanceGain, float IntegrationTimeStep, float IRSignature,
float RCS)
{

     FlightSection_EndTime = EndTime;
     FlightSection_MAXG = MAXG;
     FlightSection_SpecificImpulse = SpecificImpulse;
     FlightSection_NozzleExitArea = NozzleExitArea;
     FlightSection_ReferenceArea = ReferenceArea;
     FlightSection_InitialMass = InitialMass;
     FlightSection_MaxAlpha = MaxAlpha;
     FlightSection_ResponseTime = ResponseTime;
     FlightSection_ProNavGuidanceGain = ProNavGuidanceGain;
     FlightSection_IntegrationTimeStep = IntegrationTimeStep;
     FlightSection_IRSignature = IRSignature;
     FlightSection_RCS = RCS;
     FlightSection_extent->insert(this);
}

FlightSection::~FlightSection()
{
     FlightSection_extent->remove(this);
}

void FlightSection::FlightSection_Show(ostream& os)
{
    os << "===  EndTime: " << FlightSection_EndTime << endl;
    os << "===  MAXG: " << FlightSection_MAXG << endl;
    os << "===  LaunchElevationAngle: " << FlightSection_SpecificImpulse << endl;
    os << "===  SpecificImpulse: " << FlightSection_NozzleExitArea << endl;
    os << "===  ReferenceArea: " << FlightSection_ReferenceArea << endl;
    os << "===  InitialMass: " << FlightSection_InitialMass << endl;
    os << "===  MaxAlpha: " << FlightSection_MaxAlpha << endl;
    os << "===  ResponseTime: " << FlightSection_ResponseTime << endl;
    os << "===  ProNavGuidanceGain: " << FlightSection_ProNavGuidanceGain << endl;
    os << "===  IntegrationTimeStep: " << FlightSection_IntegrationTimeStep << endl;
    os << "===  IRSignature: " << FlightSection_IRSignature << endl;
    os << "===  RCS: " << FlightSection_RCS << endl;
}
```

Section 2. Global Classes after Suppressor integration

```
/*----------------------------------------------
--                  Airframe.cc
--
--    Airframe Implementation for Global schema
--
--               Capt Emilia Colonese
-----------------------------------------------*/

#include <string.h>
#include <stdlib.h>
#include <fstream.h>
#include <ostore/ostore.hh>
#include <ostore/coll.hh>
#include "Airframe.hh"
#include "FlightSection.hh"

Airframe::Airframe(const char* ID)
{
   strcpy(Airframe_ID, ID);

}

Airframe::~Airframe()
{
}

void Airframe::Airframe_Show(ostream& os)
{
    os << "=== Airframe ID: " << Airframe_ID << endl;
}

// constructor used for SUPRESSOR
Aircraft::Aircraft(const char* ID, const char* CapabilityName, const char* CommitAlt,
const char* FuelUsage, const char* NavErrorData, float MAXSpeed, float MINSpeed, float
MAXG, float MAXClimbAngle, float MINTurnRadius, float MINAltitude, float MAXAltitude,
float MAXDiveRate)
:Airframe(ID)
{
   strcpy(Aircraft_CapabilityName, CapabilityName);
   Aircraft_MAXSpeed = MAXSpeed;
   Aircraft_MINSpeed = MINSpeed;
   Aircraft_MAXG = MAXG;
   Aircraft_MAXClimbAngle = MAXClimbAngle;
   Aircraft_MINTurnRadius = MINTurnRadius;
   Aircraft_MINAltitude = MINAltitude;
   Aircraft_MAXAltitude = MAXAltitude;
   Aircraft_MAXDiveRate = MAXDiveRate;
   strcpy(Aircraft_CommitAlt, CommitAlt);
   strcpy(Aircraft_FuelUsage, FuelUsage);
   strcpy(Aircraft_NavErrorData, NavErrorData);

}

// constructor used for EADSIM
Aircraft::Aircraft(const char* ID, int NonAerodynamic, int MAXSpeed, int MINSpeed, float
MAXG, int EmptyWeight, int FuelWeight, int RTBFuelBingoLimit, int AirRefuelBingoLimit,
int MaxFuelReceivingRate, float LookAheadInterval, float MultiplicationFactor, int
TerrainSamples, int FeedbackControlGain, int MAXClimbAngle)
:Airframe(ID)
{
   Aircraft_NonAerodynamic = NonAerodynamic;
   Aircraft_MAXSpeed = float(MAXSpeed);
   Aircraft_MINSpeed = float(MINSpeed);
   Aircraft_MAXG = MAXG;
   Aircraft_EmptyWeight = EmptyWeight;
   Aircraft_FuelWeight = FuelWeight;
   Aircraft_RTBFuelBingoLimit = RTBFuelBingoLimit;
   Aircraft_AirRefuelBingoLimit = AirRefuelBingoLimit;
   Aircraft_MaxFuelReceivingRate = MaxFuelReceivingRate;
   Aircraft_LookAheadInterval = LookAheadInterval;
   Aircraft_MultiplicationFactor = MultiplicationFactor;
   Aircraft_TerrainSamples = TerrainSamples;
   Aircraft_FeedbackControlGain = FeedbackControlGain;
   Aircraft_MAXClimbAngle = float(MAXClimbAngle);

}

// general constructor
Aircraft::Aircraft(const char* ID, const char* CapabilityName, const char* CommitAlt,
const char* FuelUsage, const char* NavErrorData, int NonAerodynamic, float MAXSpeed,
float MINSpeed, float MAXG, int EmptyWeight, int FuelWeight, int RTBFuelBingoLimit, int
AirRefuelBingoLimit, int MaxFuelReceivingRate, float LookAheadInterval, float
MultiplicationFactor, int TerrainSamples, int FeedbackControlGain, float MAXClimbAngle,
float MINTurnRadius, float MINAltitude, float MAXAltitude, float MAXDiveRate)
:Airframe(ID)
{
   strcpy(Aircraft_CapabilityName, CapabilityName);
   Aircraft_NonAerodynamic = NonAerodynamic;
   Aircraft_MAXSpeed = MAXSpeed;
   Aircraft_MINSpeed = MINSpeed;
   Aircraft_MAXG = MAXG;
   Aircraft_EmptyWeight = EmptyWeight;
   Aircraft_FuelWeight = FuelWeight;
   Aircraft_RTBFuelBingoLimit = RTBFuelBingoLimit;
   Aircraft_AirRefuelBingoLimit = AirRefuelBingoLimit;
   Aircraft_MaxFuelReceivingRate = MaxFuelReceivingRate;
   Aircraft_LookAheadInterval = LookAheadInterval;
   Aircraft_MultiplicationFactor = MultiplicationFactor;
   Aircraft_TerrainSamples = TerrainSamples;
   Aircraft_FeedbackControlGain = FeedbackControlGain;
   Aircraft_MAXClimbAngle = MAXClimbAngle;
   Aircraft_MINTurnRadius = MINTurnRadius;
   Aircraft_MINAltitude = MINAltitude;
   Aircraft_MAXAltitude = MAXAltitude;
   Aircraft_MAXDiveRate = MAXDiveRate;
   strcpy(Aircraft_CommitAlt, CommitAlt);
   strcpy(Aircraft_FuelUsage, FuelUsage);
   strcpy(Aircraft_NavErrorData, NavErrorData);
}

Aircraft::~Aircraft()
{
}

void Aircraft::Aircraft_Show(ostream& os)
{
    Airframe::Airframe_Show(os);
    os << "=== CapabilityName: " << Aircraft_CapabilityName << endl;
    os << "=== CommitAlt: " << Aircraft_CommitAlt << endl;
    os << "=== FuelUsage: " << Aircraft_FuelUsage << endl;
    os << "=== NavErrorData: " << Aircraft_NavErrorData << endl;
    os << "===  NonAerodynamic: " << Aircraft_NonAerodynamic << endl;
    os << "===  MAXSpeed: " << Aircraft_MAXSpeed << endl;
    os << "===  MINSpeed: " << Aircraft_MINSpeed << endl;
    os << "===  MAXG: " << Aircraft_MAXG << endl;
    os << "===  EmptyWeight: " << Aircraft_EmptyWeight << endl;
    os << "===  FuelWeight: " << Aircraft_FuelWeight << endl;
    os << "===  RTBFuelBingoLimit: " << Aircraft_RTBFuelBingoLimit << endl;
    os << "===  AirRefuelBingoLimit: " << Aircraft_AirRefuelBingoLimit << endl;
    os << "===  MaxFuelReceivingRate: " << Aircraft_MaxFuelReceivingRate << endl;
    os << "===  LookAheadInterval: " << Aircraft_LookAheadInterval << endl;
    os << "===  MultiplicationFactor: " << Aircraft_MultiplicationFactor << endl;
    os << "===  TerrainSamples: " << Aircraft_TerrainSamples << endl;
    os << "===  FeedbackControlGain: " << Aircraft_FeedbackControlGain << endl;
    os << "===  MAXClimbAngle: " << Aircraft_MAXClimbAngle << endl;
    os << "===  MINTurnRadius: " << Aircraft_MINTurnRadius << endl;
    os << "===  MINAltitude: " << Aircraft_MINAltitude << endl;
    os << "===  MAXAltitude: " << Aircraft_MAXAltitude << endl;
    os << "===  MAXDiveRate: " << Aircraft_MAXDiveRate << endl;
```

```
}

// general constructor
Airplane::Airplane(const char* ID, const char* CapabilityName, const char* CommitAlt,
const char* FuelUsage, const char* NavErrorData, int NonAerodynamic, float MAXSpeed,
float MINSpeed, float MAXG, int EmptyWeight, int FuelWeight, int RTBFuelBingoLimit, int
AirRefuelBingoLimit, int MaxFuelReceivingRate, float LookAheadInterval, float
MultiplicationFactor, int TerrainSamples, int FeedbackControlGain, float MAXClimbAngle,
float MINTurnRadius, float MINAltitude, float MAXAltitude, float MAXDiveRate, int
ABSpeed, int MAXThrust, float WingArea, float WingSpan, int CruiseAltMSL, int
CruiseSpeed, int FuelFlow, float TSFC, int MAXSpeedCruiseAlt, float CD0)
:Aircraft(ID, CapabilityName, CommitAlt, FuelUsage, NavErrorData, NonAerodynamic,
MAXSpeed, MINSpeed, MAXG, EmptyWeight, FuelWeight, RTBFuelBingoLimit,
AirRefuelBingoLimit, MaxFuelReceivingRate, LookAheadInterval, MultiplicationFactor,
TerrainSamples, FeedbackControlGain, MAXClimbAngle, MINTurnRadius, MINAltitude,
MAXAltitude, MAXDiveRate)
{
   Airplane_ABSpeed = ABSpeed;
   Airplane_MAXThrust = MAXThrust;
   Airplane_WingArea = WingArea;
   Airplane_WingSpan = WingSpan;
   Airplane_CruiseAltMSL = CruiseAltMSL;
   Airplane_CruiseSpeed = CruiseSpeed;
   Airplane_TSFC = TSFC;
   Airplane_MAXSpeedCruiseAlt = MAXSpeedCruiseAlt;
   Airplane_CD0 = CD0;
   Airplane_extent->insert(this);
}

// constructor used for EADSIM
Airplane::Airplane(const char* ID, int NonAerodynamic, int MAXSpeed, int MINSpeed, float
MAXG, int EmptyWeight, int FuelWeight, int RTBFuelBingoLimit, int AirRefuelBingoLimit,
int MaxFuelReceivingRate, float LookAheadInterval, float MultiplicationFactor, int
TerrainSamples, int FeedbackControlGain, int MAXClimbAngle, int ABSpeed, int MAXThrust,
float WingArea, float WingSpan, int CruiseAltMSL, int CruiseSpeed, int FuelFlow, float
TSFC, int MAXSpeedCruiseAlt, float CD0)
:Aircraft(ID, NonAerodynamic, MAXSpeed, MINSpeed, MAXG, EmptyWeight, FuelWeight,
RTBFuelBingoLimit, AirRefuelBingoLimit, MaxFuelReceivingRate, LookAheadInterval,
MultiplicationFactor, TerrainSamples, FeedbackControlGain, MAXClimbAngle)
{
   Airplane_ABSpeed = ABSpeed;
   Airplane_MAXThrust = MAXThrust;
   Airplane_WingArea = WingArea;
   Airplane_WingSpan = WingSpan;
   Airplane_CruiseAltMSL = CruiseAltMSL;
   Airplane_CruiseSpeed = CruiseSpeed;
   Airplane_TSFC = TSFC;
   Airplane_MAXSpeedCruiseAlt = MAXSpeedCruiseAlt;
   Airplane_CD0 = CD0;
   Airplane_extent->insert(this);
}

// constructor used for SUPRESSOR
Airplane::Airplane(const char* ID, const char* CapabilityName, const char* CommitAlt,
const char* FuelUsage, const char* NavErrorData, float MAXSpeed, float MINSpeed, float
MAXG, float MAXClimbAngle, float MINTurnRadius, float MINAltitude, float MAXAltitude,
float MAXDiveRate)
:Aircraft(ID, CapabilityName, CommitAlt, FuelUsage, NavErrorData, MAXSpeed, MINSpeed,
MAXG, MAXClimbAngle, MINTurnRadius, MINAltitude, MAXAltitude, MAXDiveRate)
{
 Airplane_extent->insert(this);
}

Airplane::~Airplane()
{
   Airplane_extent->remove(this);
}

void Airplane::Airplane_Show(ostream& os)
{
    Aircraft::Aircraft_Show(os);
    os << "===  ABSpeed: " << Airplane_ABSpeed << endl;
    os << "===  MAXThrust: " << Airplane_MAXThrust << endl;
    os << "===  WingArea: " << Airplane_WingArea << endl;
    os << "===  WingSpan: " << Airplane_WingSpan << endl;
    os << "===  CruiseAltMSL: " << Airplane_CruiseAltMSL << endl;
    os << "===  CruiseSpeed: " << Airplane_CruiseSpeed << endl;
    os << "===  TSFC: " << Airplane_TSFC << endl;
    os << "===  MAXSpeedCruiseAlt: " << Airplane_MAXSpeedCruiseAlt << endl;
    os << "===  CD0: " << Airplane_CD0 << endl;
}

// general constructor
Helicopter::Helicopter(const char* ID, const char* CapabilityName, const char* CommitAlt,
const char* FuelUsage, const char* NavErrorData, int NonAerodynamic, float MAXSpeed,
float MINSpeed, float MAXG, int EmptyWeight, int FuelWeight, int RTBFuelBingoLimit, int
AirRefuelBingoLimit, int MaxFuelReceivingRate, float LookAheadInterval, float
MultiplicationFactor, int TerrainSamples, int FeedbackControlGain, float MAXClimbAngle,
float MINTurnRadius, float MINAltitude, float MAXAltitude, float MAXDiveRate, float
Power, float PSFC, float DecelarationG, int Blades, float BladeRadius, float BladeChord,
float BladeC1, float BladeCD0, float TipVelocity, float FuselageArea, float FuselageCd0)
:Aircraft(ID, CapabilityName, CommitAlt, FuelUsage, NavErrorData, NonAerodynamic,
MAXSpeed, MINSpeed, MAXG, EmptyWeight, FuelWeight, RTBFuelBingoLimit,
AirRefuelBingoLimit, MaxFuelReceivingRate, LookAheadInterval, MultiplicationFactor,
TerrainSamples, FeedbackControlGain, MAXClimbAngle, MINTurnRadius, MINAltitude,
MAXAltitude, MAXDiveRate)
{
   Helicopter_Power = Power;
   Helicopter_PSFC = PSFC;
   Helicopter_DecelarationG = DecelarationG;
   Helicopter_Blades = Blades;
   Helicopter_BladeRadius = BladeRadius;
   Helicopter_BladeChord = BladeChord;
   Helicopter_BladeC1 = BladeC1;
   Helicopter_BladeCD0 = BladeCD0;
   Helicopter_TipVelocity = TipVelocity;
   Helicopter_FuselageArea = FuselageArea;
   Helicopter_FuselageCd0 = FuselageCd0;
   Helicopter_extent->insert(this);
}

// constructor used for EADSIM
Helicopter::Helicopter(const char* ID, int NonAerodynamic, int MAXSpeed, int MINSpeed,
float MAXG, int EmptyWeight, int FuelWeight, int RTBFuelBingoLimit, int
AirRefuelBingoLimit, int MaxFuelReceivingRate, float LookAheadInterval, float
MultiplicationFactor, int TerrainSamples, int FeedbackControlGain, int MAXClimbAngle,
float Power, float PSFC, float DecelarationG, int Blades, float BladeRadius, float
BladeChord, float BladeC1, float BladeCD0, float TipVelocity, float FuselageArea, float
FuselageCd0)
:Aircraft(ID, NonAerodynamic, MAXSpeed, MINSpeed, MAXG, EmptyWeight, FuelWeight,
RTBFuelBingoLimit, AirRefuelBingoLimit, MaxFuelReceivingRate, LookAheadInterval,
MultiplicationFactor, TerrainSamples, FeedbackControlGain, MAXClimbAngle)
{
   Helicopter_Power = Power;
   Helicopter_PSFC = PSFC;
   Helicopter_DecelarationG = DecelarationG;
   Helicopter_Blades = Blades;
   Helicopter_BladeRadius = BladeRadius;
   Helicopter_BladeChord = BladeChord;
   Helicopter_BladeC1 = BladeC1;
   Helicopter_BladeCD0 = BladeCD0;
   Helicopter_TipVelocity = TipVelocity;
   Helicopter_FuselageArea = FuselageArea;
   Helicopter_FuselageCd0 = FuselageCd0;
   Helicopter_extent->insert(this);
}

// constructor used for SUPRESSOR
Helicopter::Helicopter(const char* ID, const char* CapabilityName, const char* CommitAlt,
const char* FuelUsage, const char* NavErrorData, float MAXSpeed, float MINSpeed, float
MAXG, float MAXClimbAngle, float MINTurnRadius, float MINAltitude, float MAXAltitude,
float MAXDiveRate)
```

```
:Aircraft(ID, CapabilityName, CommitAlt, FuelUsage, NavErrorData, MAXSpeed, MINSpeed,
MAXG, MAXClimbAngle, MINTurnRadius, MINAltitude, MAXAltitude, MAXDiveRate)
{
   Helicopter_extent->insert(this);
}

Helicopter::~Helicopter()
{
   Helicopter_extent->remove(this);
}

void Helicopter::Helicopter_Show(ostream& os)
{
    Aircraft::Aircraft_Show(os);
        os << "===  Power: " << Helicopter_Power << endl;
    os << "===  PSFC: " << Helicopter_PSFC << endl;
    os << "===  DecelarationG: " << Helicopter_DecelarationG << endl;
    os << "===  Blades: " << Helicopter_Blades << endl;
    os << "===  BladeRadius: " << Helicopter_BladeRadius << endl;
    os << "===  BladeChord: " << Helicopter_BladeChord << endl;
    os << "===  BladeC1: " << Helicopter_BladeC1 << endl;
    os << "===  BladeCD0: " << Helicopter_BladeCD0 << endl;
    os << "===  TipVelocity: " << Helicopter_TipVelocity << endl;
    os << "===  FuselageArea: " << Helicopter_FuselageArea << endl;
    os << "===  TipVelocity: " << Helicopter_TipVelocity << endl;
    os << "===  FuselageCd0: " << Helicopter_FuselageCd0 << endl;
}

void Missile::Set_Missile_FlightSection(FlightSection* FlighSection)
{
  Missile_FlightSections.insert(FlighSection);
}

Missile::Missile(const char* ID, float InitialVelocity, float LaunchRailLenght, float
LaunchElevationAngle, float MAXDivertVelocity, float MaxDivertLateralAcceleration)
:Airframe(ID)
{
   Missile_InitialVelocity = InitialVelocity;
   Missile_LaunchRailLenght = LaunchRailLenght;
   Missile_LaunchElevationAngle = LaunchElevationAngle;
   Missile_MAXDivertVelocity = MAXDivertVelocity;
   Missile_MaxDivertLateralAcceleration = MaxDivertLateralAcceleration;
   Missile_extent->insert(this);
}

Missile::~Missile()
{
   Missile_extent->remove(this);
}

// Show method
void Missile::Missile_Show(ostream& os)
{
    Airframe::Airframe_Show(os);
    os << "===  InitialVelocity: " << Missile_InitialVelocity << endl;
    os << "===  LaunchRailLenght: " << Missile_LaunchRailLenght << endl;
    os << "===  LaunchElevationAngle: " << Missile_LaunchElevationAngle << endl;
    os << "===  MAXDivertVelocity: " << Missile_MAXDivertVelocity << endl;
    os << "===  MaxDivertLateralAcceleration: " << Missile_MaxDivertLateralAcceleration
<< endl;
}
```

Appendix D. Data conversion programs code

Data conversion programs also called User's View translation programs. The program structure has two types of data conversion as described below.

- The first is the data conversion from the User's View to the global database.

This conversion is applied when the movement of data from the intermediate database to the global database is necessary.

- The second is the data conversion from the Global Database to the User's View.

This conversion is used to generate the specific User's View that will be translated later to the source database in order to run the related scenario generator program.

This Appendix shows the code of the User's View translation program for Suppressor.

```
/*----------------------------------------------------------------
--                              Maindcsup.cc
--
--         Implementation of the Data Conversor for Suppressor
--
--                        Capt Emilia Colonese
--
---------------------------------------------------------------*/

#include <iostream.h>
#include <fstream.h>
#include <ostore/ostore.hh>
#include <ostore/coll.hh>
#include <string.h>
#include <stdlib.h>
#include "Airframe.hh"
#include "FlightSection.hh"
#include "Mover.hh"
#include "SUP_Mover.hh"
#include "SUP_CapabilityMover.hh"

#define MAX_CHAR 200   // max input characters

    os_Set<Airplane*> *Airplane_extent = 0;
    os_Set<Helicopter*> *Helicopter_extent = 0;
    os_Set<Missile*> *Missile_extent = 0;
    os_Set<FlightSection*> *FlightSection_extent = 0;
    os_Set<MoverAirplane*> *MoverAirplane_extent = 0;
    os_Set<MoverHelicopter*> *MoverHelicopter_extent = 0;

    os_Set<SUP_MoverAirplane*> *SUP_MoverAirplane_extent = 0;
    os_Set<SUP_MoverHelicopter*> *SUP_MoverHelicopter_extent = 0;
    os_Set<SUP_CapabilityMover*> *SUP_CapabilityMover_extent = 0;

    os_typespec *Airplane_type = Airplane::get_os_typespec();
    os_typespec *Airplane_extent_type = os_Set<Airplane*>::get_os_typespec();
    os_typespec *Helicopter_type = Helicopter::get_os_typespec();
    os_typespec *Helicopter_extent_type = os_Set<Helicopter*>::get_os_typespec();
    os_typespec *Missile_type = Missile::get_os_typespec();
    os_typespec *Missile_extent_type = os_Set<Missile*>::get_os_typespec();
    os_typespec *FlightSection_type = FlightSection::get_os_typespec();
    os_typespec *FlightSection_extent_type = os_Set<FlightSection*>::get_os_typespec();
    os_typespec *MoverAirplane_type = MoverAirplane::get_os_typespec();
    os_typespec *MoverAirplane_extent_type = os_Set<MoverAirplane*>::get_os_typespec();
    os_typespec *MoverHelicopter_type = MoverHelicopter::get_os_typespec();
    os_typespec *MoverHelicopter_extent_type =
os_Set<MoverHelicopter*>::get_os_typespec();
    os_typespec *SUP_MoverAirplane_type = SUP_MoverAirplane::get_os_typespec();
    os_typespec *SUP_MoverAirplane_extent_type =
os_Set<SUP_MoverAirplane*>::get_os_typespec();
    os_typespec *SUP_MoverHelicopter_type = SUP_MoverHelicopter::get_os_typespec();
    os_typespec *SUP_MoverHelicopter_extent_type =
os_Set<SUP_MoverHelicopter*>::get_os_typespec();
    os_typespec *SUP_CapabilityMover_type = SUP_CapabilityMover::get_os_typespec();
    os_typespec *SUP_CapabilityMover_extent_type =
os_Set<SUP_CapabilityMover*>::get_os_typespec();

    os_database *db1, *db2;

void Initialize1DB1(os_database* db1)
{
    cout << endl << "Creating new classes root for semantic dictionary...";

    // Create Airplane Root
    Airplane_extent = &os_Set<Airplane*>::create(db1);
    db1->create_root("Airplane_extent_root")
->set_value(Airplane_extent, Airplane_extent_type);
    cout << ".";

    // Create Helicopter Root
    Helicopter_extent = &os_Set<Helicopter*>::create(db1);
    db1->create_root("Helicopter_extent_root")
->set_value(Helicopter_extent, Helicopter_extent_type);
    cout << ".";

    // Create Missile Root
    Missile_extent = &os_Set<Missile*>::create(db1);
    db1->create_root("Missile_extent_root")
->set_value(Missile_extent, Missile_extent_type);
    cout << ".";

    // Create FlightSection Root
    FlightSection_extent = &os_Set<FlightSection*>::create(db1);
    db1->create_root("FlightSection_extent_root")
->set_value(FlightSection_extent, FlightSection_extent_type);
    cout << ".";
}

void Initialize2DB1(os_database* db1)
{

     // Create MoverAirplane Root
    MoverAirplane_extent = &os_Set<MoverAirplane*>::create(db1);
    db1->create_root("MoverAirplane_extent_root")
->set_value(MoverAirplane_extent, MoverAirplane_extent_type);
    cout << ".";

    // Create MoverHelicopter Root
    MoverHelicopter_extent = &os_Set<MoverHelicopter*>::create(db1);
    db1->create_root("MoverHelicopter_extent_root")
->set_value(MoverHelicopter_extent, MoverHelicopter_extent_type);
    cout << ".";

}

void InitializeDB2(os_database* db2)
{
    cout << "Creating exported database for SUPRESSOR...";

    // Create SUP_MoverAirplane Root
    SUP_MoverAirplane_extent = &os_Set<SUP_MoverAirplane*>::create(db2);
    db2->create_root("SUP_MoverAirplane_extent_root")
->set_value(SUP_MoverAirplane_extent, SUP_MoverAirplane_extent_type);
    cout << ".";

    // Create SUP_MoverHelicopter Root
    SUP_MoverHelicopter_extent = &os_Set<SUP_MoverHelicopter*>::create(db2);
    db2->create_root("SUP_MoverHelicopter_extent_root")
->set_value(SUP_MoverHelicopter_extent, SUP_MoverHelicopter_extent_type);
    cout << ".";

    // Create SUP_CapabilityMover Root
    SUP_CapabilityMover_extent = &os_Set<SUP_CapabilityMover*>::create(db2);
    db2->create_root("SUP_CapabilityMover_extent_root")
->set_value(SUP_CapabilityMover_extent, SUP_CapabilityMover_extent_type);
    cout << ".";

}

void PopulateDB2(os_database* db2)
{
    cout << endl << "Populating the exported database with data examples...";
    SUP_MoverAirplane* SUP_aAux;
    SUP_MoverHelicopter* SUP_hAux;
    SUP_CapabilityMover* SUP_cAux;

    SUP_cAux = new(db2, SUP_CapabilityMover_type) SUP_CapabilityMover(
                    "Capability-F14",
                    "DIMENSION1 MT 500.0",
                    "DIMENSION2 LB 60.3",
                    "1.5 2.5 3.5 4.5 5.5 6.5 7.5 8.5",
                    1000.1,
                        200.4,
                        100.5,
                        30.4,
                        356.9,
                        34.8,
                        98.2,
                        50.8);
    cout << endl << "Capability-F14 INSERTED..";

    SUP_aAux = new(db2, SUP_MoverAirplane_type) SUP_MoverAirplane(1, "F-14", SUP_cAux);

    cout << endl << "F-14 INSERTED..";

    SUP_cAux = new(db2, SUP_CapabilityMover_type) SUP_CapabilityMover(
                        "Capability-SUPERPUMA",
                        "DIMENSION1 MT 100.0",
                        "DIMENSION2 LB 30.3",
                        "1.5 2.5 3.5 4.5 5.5 6.5 7.5 8.5",
                        500.0,
                        60.0,
                        80.0,
                        900.0,
                        6.9,
                        56.8,
                        8.0,
                        50.8);
    cout << endl << "Capability-SUPERPUMA INSERTED..";

    SUP_hAux = new(db2, SUP_MoverHelicopter_type) SUP_MoverHelicopter(2, "SUPERPUMA",
SUP_cAux);

    cout << endl << "SUPERPUMA INSERTED..";

}

int main()
{
   OS_ESTABLISH_FAULT_HANDLER
   OS_BEGIN_TXN(tx1, 0, os_transaction::update)
   // Initialize the  runtime
   cout << endl << "Initializing os ..." << endl;

   objectstore::initialize();
   os_collection::initialize();

   // Open the database named "dictionary.db".

   os_database *db1 = os_database::open("sdictionary.db", 0, 0666);

   os_database_root* dictionary_root =  db1->find_root("Airplane_extent_root");

   if (!dictionary_root)
   {
      cout << "First Initialization of db1 ..." << endl;
      Initialize1DB1(db1);
      cout << "Done." << endl;
   }

    os_database_root* dictionary_root2 =  db1->find_root("MoverAirplane_extent_root");

   if (!dictionary_root2)
   {
      cout << "Second Initialization of db1 ..." << endl;
      Initialize2DB1(db1);
      cout << "Done." << endl;
   }

   // Close database
   db1->close();
   OS_END_TXN(tx1)
   OS_END_FAULT_HANDLER
   OS_ESTABLISH_FAULT_HANDLER
   OS_BEGIN_TXN(tx2, 0, os_transaction::update)
   // Initialize the  runtime

   // Open the database named "SUPexported.db". If one does not exist, create and open
one.
   os_database *db2 = os_database::open("supexported.db", 0, 0666);
   os_database_root* SUP_root =  db2
->find_root("SUP_MoverAirplane_extent_root");

   if (!SUP_root)
    {
     cout << endl << "Initializing db2 ..." << endl;
     InitializeDB2(db2);
     cout << "Done." << endl;
    }
    else
      cout << "SUP_root found in db2 = " << &db2 << endl;

   // Close database
   db2->close();

   OS_END_TXN(tx2)
   OS_END_FAULT_HANDLER



  char next_action = ' '; // input from user

  char na[31];  // used by user action

   cout << "\nThis program convert data from the exported SUPPRESOR schema to integrate
into the Global Schema" << endl <<
           "or convert from Global Schema to the exported SUPPRESOR schema to generate
the user's view" << endl << endl;

   // Convert Class SUP_MoverAirplane

   OS_ESTABLISH_FAULT_HANDLER
   OS_BEGIN_TXN(tx3, 0, os_transaction::update)

   os_database *db2 = os_database::open("supexported.db", 0, 0666);

   os_database *db1 = os_database::open("sdictionary.db", 0, 0666);

   char choice_conversor = ' ';

   cout << "Enter with the type of conversion. " << endl
        << " Exported Schema to Global Schema (E) " << endl
        << " Global Schema to Exported Schema - View Generation (G) " << endl
        << " Quit (Q) " << endl
        << " ==> ";
   cin >> choice_conversor;

   cout << "choice ==> " << choice_conversor << endl << endl;

   // find root

   os_database_root* SUP_root =  db2->find_root("SUP_MoverAirplane_extent_root");
   if (!SUP_root)
      { cout << "sup root not found" << endl;
        return 0;}
   else
      {
        if (choice_conversor == 'E')
        {
           PopulateDB2(db2);
           cout << "Done." << endl;
        }
      }
   Airplane_extent = (os_Set<Airplane*>*) (db1
->find_root("Airplane_extent_root")
->get_value(Airplane_extent_type));

   if (!Airplane_extent)
   { cout << "Airplane root not found." << endl;
     return 0 ; }

   MoverAirplane_extent = (os_Set<MoverAirplane*>*) (db1
->find_root("MoverAirplane_extent_root")
->get_value(MoverAirplane_extent_type));

   if (!MoverAirplane_extent)
   { cout << "MoverAirplane root not found." << endl;
        return 0; }

   MoverHelicopter_extent = (os_Set<MoverHelicopter*>*) (db1
->find_root("MoverHelicopter_extent_root")
->get_value(MoverHelicopter_extent_type));

   if (!MoverHelicopter_extent)
   { cout << "MoverHelicopter root not found." << endl;
        return 0; }

   SUP_MoverAirplane_extent = (os_Set<SUP_MoverAirplane*>*) (db2
->find_root("SUP_MoverAirplane_extent_root")
->get_value(SUP_MoverAirplane_extent_type));

   if (!SUP_MoverAirplane_extent)
   { cout << "SUP_MoverAirplane root not found." << endl;
        return 0; }

   Helicopter_extent = (os_Set<Helicopter*>*) (db1
->find_root("Helicopter_extent_root")
->get_value(os_Set<Helicopter*>::get_os_typespec()));
   if (!Helicopter_extent)
   { cout << "Helicopter root not found." << endl;
     return 0 ; }

   SUP_MoverHelicopter_extent = (os_Set<SUP_MoverHelicopter*>*) (db2
->find_root("SUP_MoverHelicopter_extent_root")
->get_value(os_Set<SUP_MoverHelicopter*>::get_os_typespec()));
   if (!SUP_MoverHelicopter_extent)
   { cout << "SUP_MoverHelicopter root not found." << endl;
     return 0; }

   Missile_extent = (os_Set<Missile*>*) (db1->find_root("Missile_extent_root")
->get_value(os_Set<Missile*>::get_os_typespec()));
   if (!Missile_extent)
   { cout << "Missile root not found." << endl;
     return 0 ; }

   FlightSection_extent = (os_Set<FlightSection*>*) (db1
->find_root("FlightSection_extent_root")
->get_value(os_Set<FlightSection*>::get_os_typespec()));
   if (!FlightSection_extent)
   { cout << "Airframe root not found." << endl;
     return 0; }

   SUP_CapabilityMover_extent = (os_Set<SUP_CapabilityMover*>*) (db2
->find_root("SUP_CapabilityMover_extent_root")
->get_value(os_Set<SUP_CapabilityMover*>::get_os_typespec()));
   if (!SUP_CapabilityMover_extent)
   { cout << "SUP_CapabilityMover root not found." << endl;
     return 0; }

   switch(choice_conversor) {

     case 'E': {
        // Exported Schema to Global Schema

        SUP_MoverAirplane* SUP_aAux;
        SUP_MoverHelicopter* SUP_hAux;
        SUP_CapabilityMover* SUP_cAux;
        Airplane* aAux;
        Helicopter* hAux;
        MoverAirplane* maAux;
        MoverHelicopter* mhAux;

        os_Cursor<SUP_MoverAirplane*> c(*SUP_MoverAirplane_extent);

        for (SUP_aAux = c.first(); SUP_aAux; SUP_aAux = c.next())
        {
            SUP_cAux = SUP_aAux->SUP_Get_MoverAirplane_CapabilityAirplane();
            aAux = new(db1, Airplane::get_os_typespec()) Airplane(
                 SUP_aAux->SUP_Get_MoverAirplane_Name(),
                 SUP_cAux->SUP_Get_CapabilityMover_Name(),
                 SUP_cAux->SUP_Get_CapabilityMover_CommitAlt(),
                 SUP_cAux->SUP_Get_CapabilityMover_FuelUsage(),
                 SUP_cAux->SUP_Get_CapabilityMover_NavErrorData(),
                 SUP_cAux->SUP_Get_CapabilityMover_MAXSpeed(),
                 SUP_cAux->SUP_Get_CapabilityMover_MINSpeed(),
                 SUP_cAux->SUP_Get_CapabilityMover_MAXAcceleration(),
                 SUP_cAux->SUP_Get_CapabilityMover_MAXClimbRate(),
                 SUP_cAux->SUP_Get_CapabilityMover_MINTurnRadius(),
                 SUP_cAux->SUP_Get_CapabilityMover_MINAltitude(),
                 SUP_cAux->SUP_Get_CapabilityMover_MAXAltitude(),
                 SUP_cAux->SUP_Get_CapabilityMover_MAXDiveRate());

            cout << "Airplane instance created " << endl;

            maAux = new(db1, MoverAirplane_type) MoverAirplane(SUP_aAux
->SUP_Get_MoverAirplane_SystemID(),
                                                              aAux);

            cout << "SUP_CapabilityMover data for an SUP_MoverAirplane instance transfered
" << endl;
        }
        cout << endl << "SUP_MoverAirplane data transfered " << endl;

        os_Cursor<SUP_MoverHelicopter*> d(*SUP_MoverHelicopter_extent);
        for (SUP_hAux = d.first(); SUP_hAux; SUP_hAux = d.next())
        {
          SUP_cAux = SUP_hAux->SUP_Get_MoverHelicopter_CapabilityHelicopter();
          hAux = new(db1, Helicopter::get_os_typespec()) Helicopter(
             SUP_hAux->SUP_Get_MoverHelicopter_Name(),
             SUP_cAux->SUP_Get_CapabilityMover_Name(),
             SUP_cAux->SUP_Get_CapabilityMover_CommitAlt(),
             SUP_cAux->SUP_Get_CapabilityMover_FuelUsage(),
             SUP_cAux->SUP_Get_CapabilityMover_NavErrorData(),
             SUP_cAux->SUP_Get_CapabilityMover_MAXSpeed(),
             SUP_cAux->SUP_Get_CapabilityMover_MINSpeed(),
             SUP_cAux->SUP_Get_CapabilityMover_MAXAcceleration(),
             SUP_cAux->SUP_Get_CapabilityMover_MAXClimbRate(),
             SUP_cAux->SUP_Get_CapabilityMover_MINTurnRadius(),
             SUP_cAux->SUP_Get_CapabilityMover_MINAltitude(),
             SUP_cAux->SUP_Get_CapabilityMover_MAXAltitude(),
             SUP_cAux->SUP_Get_CapabilityMover_MAXDiveRate());

            mhAux = new(db1, MoverHelicopter::get_os_typespec()) MoverHelicopter(SUP_hAux
->SUP_Get_MoverHelicopter_SystemID(), hAux);

            cout << "SUP_CapabilityMover data for an SUP_MoverHelicopter instance
transfered " << endl;
        }
        cout << "SUP_MoverHelicopter data transfered " << endl;

        break;
        }
```

```
      case 'G': {

      // Global Schema to Exported Schema

      Airplane* aAux;
      Helicopter* hAux;
      MoverAirplane* maAux;
      MoverHelicopter* mhAux;
      SUP_MoverAirplane* SUP_aAux;
      SUP_MoverHelicopter* SUP_hAux;
      SUP_CapabilityMover* SUP_cAux;

      os_Cursor<MoverAirplane*> c(*MoverAirplane_extent);
      for (maAux = c.first(); maAux; maAux = c.next())
      {
        aAux = maAux->Get_MoverAirplane_CapabilityAirplane();
        SUP_cAux = new(db2, SUP_CapabilityMover::get_os_typespec())
SUP_CapabilityMover(aAux->Get_Airplane_CapabilityName(),
                    aAux->Get_Airplane_CommitAlt(),
                    aAux->Get_Airplane_FuelUsage(),
                    aAux->Get_Airplane_NavErrorData(),
                    aAux->Get_Airplane_MAXG(),
                    aAux->Get_Airplane_MINAltitude(),
                    aAux->Get_Airplane_MAXAltitude(),
                    aAux->Get_Airplane_MINTurnRadius(),
                    aAux->Get_Airplane_MINSpeed(),
                    aAux->Get_Airplane_MAXSpeed(),
                    aAux->Get_Airplane_MAXDiveRate(),
                    aAux->Get_Airplane_MAXClimbAngle());

        cout << endl << "Airplane and MoverAirplane data transfered to
SUP_CapabilityMover" << endl;


        SUP_aAux = new (db2, SUP_MoverAirplane::get_os_typespec())
SUP_MoverAirplane(maAux
->Get_MoverAirplane_SystemID(),

maAux->Get_MoverAirplane_Name(),

SUP_cAux);
      }
      cout << endl << "Airplane and MoverAirplane data transfered to SUP_MoverAirplane"
<< endl;

      os_Cursor<MoverHelicopter*> d(*MoverHelicopter_extent);
      for (mhAux = d.first(); mhAux; mhAux = d.next())
      {
        hAux = mhAux
->Get_MoverHelicopter_CapabilityHelicopter();
        SUP_cAux = new(db2, SUP_CapabilityMover::get_os_typespec())
SUP_CapabilityMover(hAux->Get_Helicopter_CapabilityName(),
       hAux->Get_Helicopter_CommitAlt(),
       hAux->Get_Helicopter_FuelUsage(),
       hAux->Get_Helicopter_NavErrorData(),
       hAux->Get_Helicopter_MAXG(),
       hAux->Get_Helicopter_MINAltitude(),
       hAux->Get_Helicopter_MAXAltitude(),
       hAux->Get_Helicopter_MINTurnRadius(),
       hAux->Get_Helicopter_MINSpeed(),
       hAux->Get_Helicopter_MAXSpeed(),
       hAux->Get_Helicopter_MAXDiveRate(),
       hAux->Get_Helicopter_MAXClimbAngle());

        cout << endl << "Helicopter and MoverHelicopter data transfered to
SUP_CapabilityMover" << endl;

        SUP_hAux = new(db2, SUP_MoverHelicopter::get_os_typespec())
SUP_MoverHelicopter(mhAux->Get_MoverHelicopter_SystemID(),

mhAux->Get_MoverHelicopter_Name(),
SUP_cAux);
      }
      cout << endl << "Helicopter and MoverHelicopter data transfered to
SUP_MoverHelicopter" << endl;

      break;
      }

     case 'Q': break;

     }
     db1->close();
     db2->close();
     OS_END_TXN(tx3)
     OS_END_FAULT_HANDLER
}
```

Appendix E. Data Verification program code

```
/*-------------------------------------------------------
--                          Maingd.cc
--
--          Implementation of the Global Data Access
--
--                    Capt Emilia Colonese
--
-------------------------------------------------------*/

#include <iostream.h>
#include <fstream.h>
#include <ostore/ostore.hh>
#include <ostore/coll.hh>
#include <string.h>
#include <stdlib.h>
#include "Airframe.hh"
#include "FlightSection.hh"
#include "Mover.hh"

os_Set<Aircraft*> *Aircraft_extent = 0;
os_Set<Airplane*> *Airplane_extent = 0;
os_Set<Helicopter*> *Helicopter_extent = 0;
os_Set<Missile*> *Missile_extent = 0;
os_Set<FlightSection*> *FlightSection_extent = 0;
os_Set<MoverAirplane*> *MoverAirplane_extent = 0;
os_Set<MoverHelicopter*> *MoverHelicopter_extent = 0;

os_typespec *MoverAirplane_type = MoverAirplane::get_os_typespec();
os_typespec *MoverAirplane_extent_type = os_Set<MoverAirplane*>::get_os_typespec();
os_typespec *MoverHelicopter_type = MoverHelicopter::get_os_typespec();
os_typespec *MoverHelicopter_extent_type = os_Set<MoverHelicopter*>::get_os_typespec();

os_typespec *Airplane_type = Airplane::get_os_typespec();
os_typespec *Airplane_extent_type = os_Set<Airplane*>::get_os_typespec();
os_typespec *Helicopter_type = Helicopter::get_os_typespec();
os_typespec *Helicopter_extent_type = os_Set<Helicopter*>::get_os_typespec();
os_typespec *Missile_type = Missile::get_os_typespec();
os_typespec *Missile_extent_type = os_Set<Missile*>::get_os_typespec();
os_typespec *FlightSection_type = FlightSection::get_os_typespec();
os_typespec *FlightSection_extent_type = os_Set<FlightSection*>::get_os_typespec();

os_database *db;

int main()
{

        OS_ESTABLISH_FAULT_HANDLER
        OS_BEGIN_TXN(tx1, 0, os_transaction::update)

        // Initialize the  runtime
        cout << endl << "Initializing os ..." << endl;
        objectstore::initialize();
        os_collection::initialize();

        os_database *db = os_database::open("sdictionary.db", 0, 0666);

        ofstream OutputFile;

        char auxname[51];
        cout << "Class Name?" << endl << "==> ";
        cin.getline(auxname, 50);

        cout << endl << "Printing data for class ==> " << auxname;
        if (strlen(auxname)==0)
        {
          cout << "Class name invalid! " << endl;
          return 0;
        }

// these classes are not concrete, threfore are not part of the implemented classes
/*        if (strcmp(auxname, "Airframe")==0)
```

```
        {
            OutputFile.open("Airframedat.txt");
            OutputFile << endl << "Data for Class Airframe " << endl;
            Airframe_extent = (os_Set<Airframe*>*) (db
->find_root("Airframe_extent_root")->get_value(Airplane_extent_type));

            if (!Airframe_extent)
            { cout << "Airframe root not found." << endl;
              return 0;
            }

            os_Cursor<Airframe*> a(*Airframe_extent);
            Airframe* aAux;

            if (a.first())
            {  for (aAux = a.first(); aAux; aAux = a.next())
                 {  aAux->Airframe_Show(OutputFile);
                    OutputFile << endl;
                 }
            }
            else
               cout << "no Airframe data stored!" << endl;

        }

        if (strcmp(auxname, "Aircraft")==0)
        {

            OutputFile.open("Aircraftdat.txt");
            OutputFile << endl << "Data for Class Aircraft: " << endl;

            Aircraft_extent = (os_Set<Aircraft*>*) (db
->find_root("Aircraft_extent_root")
->get_value(Airplane_extent_type));

            if (!Aircraft_extent)
            { cout << "Aircraft root not found." << endl;
              return 0;
            }

            os_Cursor<Aircraft*> b(*Aircraft_extent);
            Aircraft* bAux;

            if (b.first())
            {  for (bAux = b.first(); bAux; bAux = b.next())
               {  bAux->Aircraft_Show(OutputFile);
                  OutputFile << endl;
               }
            }
             else
               cout << "no Aircraft data stored!" << endl;
        }
*/

        if (strcmp(auxname, "Airplane")==0)
        {
            OutputFile.open("Airplanedat.txt");
            OutputFile << endl << "Data for Class Airplane: " << endl;

            Airplane_extent = (os_Set<Airplane*>*) (db
->find_root("Airplane_extent_root")
->get_value(Airplane_extent_type));

            if (!Airplane_extent)
            { cout << "Airplane root not found." << endl;
              return 0;
            }

            os_Cursor<Airplane*> c(*Airplane_extent);
            Airplane* cAux;
            if (c.first())
            {  for (cAux = c.first(); cAux; cAux = c.next())
                {  cAux->Airplane_Show(OutputFile);
                   OutputFile << endl;
                }
            }
            else
                cout << "no Airplane data stored!" << endl;

        }

        if (strcmp(auxname, "Helicopter")==0)
        {
           OutputFile.open("Helicopterdat.txt");
           OutputFile << endl << "Data for Class Helicopter: " << endl;

           Helicopter_extent = (os_Set<Helicopter*>*) (db
->find_root("Helicopter_extent_root")
->get_value(os_Set<Helicopter*>::get_os_typespec()));
           if (!Helicopter_extent)
           { cout << "Helicopter root not found." << endl;
             return 0 ; }

           os_Cursor<Helicopter*> d(*Helicopter_extent);
           Helicopter* dAux;

           if (d.first())
           {  for (dAux = d.first(); dAux; dAux = d.next())
               {       dAux->Helicopter_Show(OutputFile);
                 OutputFile << endl;
               }
           }
           else
               cout << "no Helicopter data stored!" << endl;

        }

        if (strcmp(auxname, "MoverAirplane")==0)
        {
           OutputFile.open("MoverAirplanedat.txt");
           OutputFile << endl << "Data for Class MoverAirplane: " << endl;

           MoverAirplane_extent = (os_Set<MoverAirplane*>*) (db
->find_root("MoverAirplane_extent_root")
->get_value(os_Set<MoverAirplane*>::get_os_typespec()));
           if (!MoverAirplane_extent)
           { cout << "MoverAirplane root not found." << endl;
             return 0; }

           os_Cursor<MoverAirplane*> g(*MoverAirplane_extent);
           MoverAirplane* gAux;

           if (g.first())
           {
               for (gAux = g.first(); gAux; gAux = g.next())
               {
                 gAux->MoverAirplane_Show(OutputFile);

                 Airplane* hAux;
                 hAux = gAux
->Get_MoverAirplane_CapabilityAirplane();
                 if (hAux)
                {
                   OutputFile << endl << "CapabilityAirplane Data for this MoverAirplane:
" << endl;
                   hAux->Airplane_Show(OutputFile);
                   OutputFile << endl;

                 }
                 else
```

```
                    cout << "no Airplane data stored for this MoverAirplane!" << endl <<
endl;
                }
            }
            else
                cout << "no MoverAirplane data stored!" << endl;

        }

        if (strcmp(auxname, "MoverHelicopter")==0)
        {
            OutputFile.open("MoverHelicopterdat.txt");
            OutputFile << endl << "Data for Class MoverHelicopter: " << endl;

            MoverHelicopter_extent = (os_Set<MoverHelicopter*>*) (db
->find_root("MoverHelicopter_extent_root")
->get_value(os_Set<MoverHelicopter*>::get_os_typespec()));
            if (!MoverHelicopter_extent)
            { cout << "MoverHelicopter root not found." << endl;
              return 0; }

            os_Cursor<MoverHelicopter*> i(*MoverHelicopter_extent);
            MoverHelicopter* iAux;

            if (i.first())
            {
               for (iAux = i.first(); iAux; iAux = i.next())
               {
                 iAux->MoverHelicopter_Show(OutputFile);

                 Helicopter* jAux;
                 jAux = iAux
->Get_MoverHelicopter_CapabilityHelicopter();

                 if (jAux)
                {
                    OutputFile << endl << "CapabilityHelicopter Data for this
MoverHelicopter: " << endl;
                   jAux->Helicopter_Show(OutputFile);
                   OutputFile << endl;

                 }
                 else
                    cout << "no Helicopter data stored for this MoverHelicopter!" << endl
<< endl;
               }
            }
            else
                cout << "no MoverHelicopter data stored!" << endl;

        }

        if (strcmp(auxname, "Missile")==0)
        {
            OutputFile.open("Missiledat.txt");
            OutputFile << endl << "Data for Class Missile: " << endl;

            Missile_extent = (os_Set<Missile*>*) (db
->find_root("Missile_extent_root")
->get_value(os_Set<Missile*>::get_os_typespec()));
            if (!Missile_extent)
            { cout << "Missile root not found." << endl;
              return 0; }

            os_Cursor<Missile*> e(*Missile_extent);
            Missile* eAux;

            if (e.first())
            {
               for (eAux = e.first(); eAux; eAux = e.next())
               {
                eAux->Missile_Show(OutputFile);

                os_Cursor<FlightSection*> f(eAux
->Get_Missile_FlightSections());
                FlightSection* fAux;
                if (f.first())
               {
                  OutputFile << endl << "FlightSection Data for this Missile: " << endl;
                  for (fAux = f.first(); fAux; fAux = f.next())
                  {    fAux->FlightSection_Show(OutputFile);
                       OutputFile << endl;
                   }
                }
                else
                   cout << "no FlightSections data stored for this Missile!" << endl <<
endl;
              }
           }
           else
               cout << "no Missile data stored!" << endl;

        }
        cout << " ...Done " << endl;
        db->close();
        OutputFile.close();

        OS_END_TXN(tx1)
        OS_END_FAULT_HANDLER
}
```

Vita

Captain Emília de Menezes Colonese was born on 6 July 1964 in Rio de Janeiro, Brazil. In 1983 she joined the Brazilian Air Force and began work as a computer programmer. She received the Bachelor of Computer Systems degree in 1984 from the Faculdade de Adminstração da Guanabara (FAG) in Rio de Janeiro. She then continued her work for the Air Force as a computer systems analyst. In 1991 she received a degree in Business Administration from FAG. In June 1997 she was assigned to the Air Force Institute of Technology at Wright-Patterson AFB, Ohio to pursue a Master of Science degree in Computer Systems with an emphasis in Database Systems.

Permanent address: Rua Souza Lima, 422 apto 502
Copacabana, Rio de Janeiro – Brazil
CEP 22081

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** JUNE 1999	**3. REPORT TYPE AND DATES COVERED**

4. TITLE AND SUBTITLE Methodology for Integrating the Scenario Databases of Simulation Systems	**5. FUNDING NUMBERS**
6. AUTHOR(S) Colonese, E.M.	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology (AFIT) Building 640 2950 P Street, WPFAB OH 45433-7765	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/GCS/ENG/99J-03
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Foster, R.M. Air Force Reasearch Laboratory (AFRL/SNZW) Building 620 2241 Avionics Circle suite 1, WPAFB OH 45433-7303	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION AVAILABILITY STATEMENT	**12b. DISTRIBUTION CODE**

13. ABSTRACT *(Maximum 200 words)*

The use of many different simulation systems by the United States Department of Defense has resulted in many different scenario data representations contained in heterogeneous databases. These heterogeneous databases all represent the same data concept, but have different semantics due to intrinsic variations among the data models. In this research, I describe a unified scenario database to allow interoperability and reuse of the scenario data components while avoiding the problems of data redundancy. Using the object-oriented approach, the data and schema of the scenario databases, represented in an object-oriented model, are integrated into a global database also represented in an object-oriented model. The global database schema is extended to allow semantic interoperability of database components by explicitly associating the semantics of the schema elements of the database components with the global metadata. I create the Integration Dictionary to represent the semantic interoperability and to store the translation mappings between each database component and the global database. The Integration Dictionary also provides support for object-oriented views generation. Next, I describe a methodology to integrate databases using the Integration Dictionary. My methodology is based on an analysis of the semantics of conceptual schema elements and on the identification of related elements in the global schema. My methodology defines the resolution for schema conflicts, and associates these conflict resolutions with data changes in the Integration Dictionary. Selected parts of the Extended Air Defense Simulation (EADSIM) and Suppressor scenario databases are integrated into the global database to validate my methodology. I use the Object-Store database to implement the global database.

14. SUBJECT TERMS Simulations, Object-oriented, Modeling, Database, Methodology, Integration, Suppressor, EADSIM, Views, ObjectStore, C++, Interoperability, Reuse			**15. NUMBER OF PAGES** 177
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	**18. SECURITY CLASSIFICATION OF THIS PAGE** UNCLASSIFIED	**19. SECURITY CLASSIFICATION OF ABSTRACT** UNCLASSIFIED	**20. LIMITATION OF ABSTRACT** UL